THÈSE EN COTUTELLE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR

DE L'UNIVERSITÉ DE BORDEAUX

ET DE L'UNIVERSIDADE FEDERAL FLUMINENSE

ÉCOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

SPÉCIALITÉ : INFORMATIQUE

Par **Alan LIRA NUNES**

Algorithmes d'ordonnancement pour l'optimisation des modèles d'apprentissage automatique distribués sur des ressources hétérogènes

Scheduling algorithms for the optimization of distributed machine learning models on heterogeneous resources

Sous la direction de : **Laércio LIMA PILLA**
Co-directrice : **Lúcia DRUMMOND**

Soutenue le 7 août 2026

Membres du jury :

M. Laércio LIMA PILLA	Chargé de Recherche	CNRS	Directeur
Mme Lúcia DRUMMOND	Professeure des Universités	Universidade Federal Fluminense	Co-Directrice
M. Shadi IBRAHIM	Chargé de Recherche	Inria	Rapporteur
M. Alfredo GOLDMAN	Professeur des Universités	Universidade de São Paulo	Rapporteur (et Président)
M. Valmir BARBOSA	Professeur des Universités	Universidade do Estado do Rio de Janeiro	Examinateur
M. César DE ROSE	Professeur des Universités	Pontifícia Universidade Católica do Rio Grande do Sul	Examinateur
Mme Flávia DELICATO	Professeure des Universités	Universidade Federal Fluminense	Examinatrice
Mme Cristina BOERES	Professeure des Universités	Universidade Federal Fluminense	Invitée
Mme Débora SAADE	Professeure des Universités	Universidade Federal Fluminense	Invitée

UNIVERSIDADE FEDERAL FLUMINENSE
UNIVERSITÉ DE BORDEAUX

ALAN LIRA NUNES

SCHEDULING ALGORITHMS FOR THE OPTIMIZATION OF DISTRIBUTED MACHINE LEARNING MODELS ON HETEROGENEOUS RESOURCES

NITERÓI

2026

ALAN LIRA NUNES

SCHEDULING ALGORITHMS FOR THE OPTIMIZATION OF DISTRIBUTED MACHINE LEARNING MODELS ON HETEROGENEOUS RESOURCES

Thesis presented under joint supervision (cotutelle) between Universidade Federal Fluminense and Université de Bordeaux. Thesis submitted to the Graduate Program in Computing at Universidade Federal Fluminense and to the École Doctorale Mathématiques et Informatique of Université de Bordeaux in partial fulfillment of the requirements for the degree of Doctor in Computing and the *diplôme national de doctorat*. Concentration area: Computer Science.

Advisor:
LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND

Co-advisor:
MARIA CRISTINA SILVA BOERES
LAÉRCIO LIMA PILLA

NITERÓI

2026

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

N972s Nunes, Alan Lira
Scheduling algorithms for the optimization of distributed machine learning models on heterogeneous resources / Alan Lira Nunes. - 2026.
266 f.: il.

Orientador: Lúcia Maria de Assumpção Drummond.
Coorientador: Maria Cristina Silva Boeres; Laércio Lima Pilla.
Tese (doutorado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2026.

1. Aprendizado de máquina. 2. Sistemas paralelos e distribuídos. 3. Escalonamento de tarefa. 4. Otimização combinatória (Computação). 5. Produção intelectual. I. Drummond, Lúcia Maria de Assumpção, orientadora. II. Boeres, Maria Cristina Silva, coorientadora. III. Pilla, Laércio Lima, coorientador. IV. Universidade Federal Fluminense. Instituto de Computação. V. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

ALAN LIRA NUNES

SCHEDULING ALGORITHMS FOR THE OPTIMIZATION OF DISTRIBUTED MACHINE LEARNING MODELS ON HETEROGENEOUS RESOURCES

Approved on August 7, 2026.

EXAMINATION BOARD

Prof. Dr. Lúcia Maria de Assumpção Drummond – Advisor, UFF

Prof. Dr. Maria Cristina Silva Boeres – Co-advisor, UFF

Dr. Laércio Lima Pilla – Co-advisor, CNRS

Dr. Shadi Ibrahim, Inria

Prof. Dr. Alfredo Goldman Vel Lejbman, USP

Prof. Dr. Valmir Carneiro Barbosa, UERJ

Prof. Dr. César Augusto Fonticielha De Rose, PUCRS

Prof. Dr. Débora Christina Muchaluat Saade, UFF

Prof. Dr. Flávia Coimbra Delicato, UFF

Niterói

2026

Algoritmos de escalonamento para a otimização de modelos de aprendizado de máquina distribuídos em recursos heterogêneos

Resumo: O Aprendizado Federado (*Federated Learning*, FL) permite que múltiplos clientes treinem colaborativamente modelos de aprendizado de máquina sem compartilhar seus dados brutos. No entanto, o FL *cross-device* permanece desafiador na prática, pois os clientes diferem em capacidade computacional, qualidade de comunicação, energia disponível, distribuições locais de dados e confiabilidade. No FL síncrono, essas diferenças afetam a eficiência do sistema e o desempenho do aprendizado: clientes lentos, com restrições de energia ou pouco representativos podem atrasar o treinamento, aumentar o consumo de recursos ou degradar a convergência. A maioria das estratégias existentes decide apenas se um cliente participa de uma rodada de comunicação, enquanto clientes selecionados geralmente treinam o modelo sobre todos os seus dados locais, limitando a adaptação da carga por cliente. Esta tese aborda a seleção de clientes no FL síncrono *cross-device* sob a perspectiva do escalonamento de tarefas. Os clientes são modelados como recursos heterogêneos, e a carga de uma rodada é decomposta em tarefas correspondentes a subconjuntos de dados locais. Assim, o servidor pode decidir quais clientes participam e quantos dados cada um deve processar. Essa formulação permite uma alocação fina da carga sobre recursos heterogêneos e possibilita a otimização conjunta de objetivos relacionados ao sistema e ao aprendizado. Esta tese propõe inicialmente MEC e ECMTC, dois algoritmos ótimos de escalonamento para alocação de carga considerando tempo e energia. MEC minimiza primeiro a duração da rodada e depois o consumo de energia, enquanto ECMTC minimiza primeiro o consumo de energia e depois a duração da rodada sob restrição de tempo. Esses algoritmos se baseiam em programação dinâmica e fornecem escalonamentos ótimos para suas respectivas ordens de objetivos. A tese introduz então o MetaCS-FL, um *framework* baseado em meta-heurística que estende essa perspectiva para um cenário multiobjetivo mais abrangente. Além do tempo de execução e da energia, o MetaCS-FL considera a utilidade do modelo, a qualidade da distribuição de classes, a diversidade de participação e a confiabilidade dos clientes. Ele usa o escalonamento produzido pelo ECMTC como solução inicial e refina as atribuições de tarefas aos clientes por meio da meta-heurística *Large Neighborhood Search*. O *framework* também integra resseleção orientada por eventos, reúso de soluções, controle de capacidade sensível à confiabilidade e privacidade diferencial, permitindo explorar informações ruidosas sobre distribuições de classes sem acessar distribuições exatas. Os métodos propostos são avaliados em classificação de imagens e textos, sob distribuições IID e não-IID, usando clientes heterogêneos emulados e comparações com FedAvg e estratégias do estado da arte. Nos cenários avaliados, o MetaCS-FL supera globalmente os demais algoritmos e alcança o melhor compromisso entre tempo de treinamento, energia, velocidade de convergência e equidade. Sob disponibilidade estática, ele reduz o tempo total de treinamento e o consumo de energia ao atingir a acurácia-alvo em menos rodadas, sem concentrar a carga em poucos clientes. A variante privada permanece próxima da versão não privada, indicando que a divulgação ruidosa das distribuições preserva informações úteis para a seleção enquanto melhora a privacidade. Sob disponibilidade dinâmica, com clientes tardios ou intermitentes, o MetaCS-FL mantém desempenho superior ao adaptar as atribuições de carga conforme disponibilidade, conclusão e confiabilidade dos clientes.

Palavras-chave: Computação de alto desempenho, Escalonamento, Algoritmos, Aprendizado de máquina, Otimização

Algorithmes d'ordonnancement pour l'optimisation des modèles d'apprentissage automatique distribués sur des ressources hétérogènes

Résumé : L'apprentissage fédéré (*Federated Learning*, FL) permet à plusieurs clients d'entraîner collaborativement des modèles d'apprentissage automatique sans partager leurs données brutes. Cependant, le FL *cross-device* reste difficile en pratique, car les clients diffèrent en capacité de calcul, qualité de communication, énergie disponible, distributions locales de données et fiabilité. Dans le FL synchrone, ces différences affectent l'efficacité du système et les performances d'apprentissage : des clients lents, contraints en énergie ou peu représentatifs peuvent retarder l'entraînement, accroître la consommation de ressources ou dégrader la convergence. La plupart des stratégies existantes décident seulement si un client participe à un tour de communication, tandis que les clients sélectionnés entraînent généralement le modèle sur l'ensemble de leurs données locales, limitant l'adaptation de la charge par client. Cette thèse aborde la sélection de clients dans le FL synchrone *cross-device* sous l'angle de l'ordonnancement de tâches. Les clients sont modélisés comme des ressources hétérogènes, et la charge d'un tour est décomposée en tâches correspondant à des sous-ensembles de données locales. Ainsi, le serveur peut décider quels clients participent et quelle quantité de données chacun doit traiter. Cette formulation permet une allocation fine de la charge sur des ressources hétérogènes et soutient l'optimisation conjointe d'objectifs liés au système et à l'apprentissage. Cette thèse propose d'abord MEC et ECMTC, deux algorithmes d'ordonnancement optimaux pour l'allocation de charge tenant compte du temps et de l'énergie. MEC minimise d'abord la durée du tour, puis la consommation d'énergie, tandis qu'ECMTC minimise d'abord la consommation d'énergie, puis la durée du tour sous contrainte de temps. Ces algorithmes reposent sur la programmation dynamique et fournissent des ordonnancements optimaux pour leurs ordres d'objectifs respectifs. La thèse introduit ensuite MetaCS-FL, un *framework* fondé sur une métaheuristique qui étend cette perspective à un cadre multi-objectifs plus large. Au-delà du temps d'exécution et de l'énergie, MetaCS-FL prend en compte l'utilité du modèle, la qualité de la distribution des classes, la diversité de participation et la fiabilité des clients. Il utilise l'ordonnancement produit par ECMTC comme solution initiale et affine les affectations de tâches aux clients au moyen de la métaheuristique *Large Neighborhood Search*. Le *framework* intègre également une resélection déclenchée par événements, la réutilisation de solutions, un contrôle de capacité tenant compte de la fiabilité et la confidentialité différentielle, permettant d'exploiter des informations bruitées sur les distributions de classes sans accéder aux distributions exactes. Les méthodes proposées sont évaluées en classification d'images et de textes, sous distributions IID et non-IID, avec des clients hétérogènes émulés et des comparaisons avec FedAvg et des stratégies de l'état de l'art. Dans les scénarios évalués, MetaCS-FL surpasse globalement les autres algorithmes et obtient le meilleur compromis entre temps d'entraînement, énergie, vitesse de convergence et équité. En disponibilité statique, il réduit le temps total d'entraînement et la consommation d'énergie tout en atteignant la précision cible en moins de tours, sans concentrer la charge sur peu de clients. La variante privée reste proche de la version non privée, indiquant que la divulgation bruitée des distributions préserve des informations utiles à la sélection tout en améliorant la confidentialité. En disponibilité dynamique, avec des clients tardifs ou intermittents, MetaCS-FL maintient des performances supérieures en adaptant les affectations de charge selon la disponibilité, le comportement d'achèvement et la fiabilité des clients.

Mots-clés : Calcul haute performance, Ordonnancement, Algorithmique, Apprentissage automatique, Optimisation

Scheduling algorithms for the optimization of distributed machine learning models on heterogeneous resources

Abstract: Federated Learning (FL) enables multiple clients to collaboratively train machine learning models without sharing their raw data. However, practical cross-device FL remains challenging because clients differ in computational capacity, communication quality, available energy, local data distributions, and reliability. In synchronous FL, these differences affect system efficiency and learning performance: slow, energy-constrained, or poorly representative clients can delay training, increase resource consumption, or degrade convergence. Most existing strategies only decide whether a client participates in a communication round, while selected clients usually train the model on all their local data, limiting per-client workload adaptation. This thesis addresses client selection in synchronous cross-device FL from a task scheduling perspective. Clients are modeled as heterogeneous resources, and the workload of a round is decomposed into tasks corresponding to subsets of local data. Thus, the server can decide which clients participate and how much data each one should process. This formulation enables fine-grained workload allocation over heterogeneous resources and supports the joint optimization of system-level and learning-related objectives. This thesis first proposes MEC and ECMTC, two optimal scheduling algorithms for workload allocation considering time and energy. MEC first minimizes round duration and then energy consumption, whereas ECMTC first minimizes energy consumption and then round duration under a time constraint. These algorithms rely on dynamic programming and provide optimal schedules for their respective objective orderings. The thesis then introduces MetaCS-FL, a metaheuristic-based framework that extends this perspective to a broader multi-objective setting. Beyond execution time and energy, MetaCS-FL considers model utility, class-distribution quality, participation diversity, and client reliability. It uses the schedule produced by ECMTC as an initial solution and refines task assignments to clients through the Large Neighborhood Search metaheuristic. The framework also integrates event-driven reselection, solution reuse, reliability-aware capacity control, and differential privacy, allowing noisy information about class distributions to be exploited without accessing exact distributions. The proposed methods are evaluated in image and text classification, under IID and non-IID distributions, using heterogeneous emulated clients and comparisons with FedAvg and state-of-the-art strategies. In the evaluated scenarios, MetaCS-FL outperforms the other algorithms overall and achieves the best trade-off among training time, energy, convergence speed, and fairness. Under static availability, it reduces total training time and energy consumption while reaching the target accuracy in fewer rounds, without concentrating the workload on few clients. The private variant remains close to the non-private version, indicating that noisy disclosure of distributions preserves useful selection information while improving privacy. Under dynamic availability, with late-joining or intermittent clients, MetaCS-FL maintains superior performance by adapting workload assignments according to client availability, completion behavior, and reliability.

Keywords: High performance computing, Scheduling, Algorithms, Machine learning, Optimization

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
UMR CNRS 5800 Université de Bordeaux, 33400 Talence, France.

Résumé substantiel en français

Contexte de la recherche. L'apprentissage fédéré, ou *Federated Learning* (FL), est un paradigme d'apprentissage automatique distribué dans lequel plusieurs participants entraînent collectivement un modèle sans transmettre directement leurs données brutes à un serveur central. Dans un cadre classique, le serveur envoie l'état courant du modèle à un sous-ensemble de clients. Chaque client réalise un entraînement local sur ses propres données, puis le serveur agrège les mises à jour reçues afin de produire un nouveau modèle global. Ce fonctionnement répond à des préoccupations importantes liées à la confidentialité, à la souveraineté des données et aux contraintes réglementaires, car les données restent localement stockées chez les participants. Il est particulièrement pertinent lorsque les données sont sensibles, volumineuses, distribuées sur un grand nombre d'appareils, ou difficiles à centraliser.

Cependant, le passage de l'apprentissage fédéré d'un principe théorique à un système opérationnel soulève de nombreuses difficultés. Ces difficultés sont particulièrement marquées dans les scénarios dits *cross-device*, où les clients peuvent être des appareils mobiles, des objets connectés, des machines personnelles ou des dispositifs de périphérie (*edge*). Dans ce contexte, le nombre de clients potentiels peut être très élevé, mais seule une fraction d'entre eux est généralement disponible à un instant donné. Les clients peuvent différer fortement par leur puissance de calcul, leur capacité énergétique, leur qualité de connexion réseau, leur disponibilité temporelle et la quantité ou la nature des données qu'ils possèdent. De plus, leurs données sont souvent non indépendantes et identiquement distribuées, c'est-à-dire non-IID : certains clients peuvent contenir uniquement quelques classes, certains profils de données ou certains comportements locaux. Cette hétérogénéité statistique peut ralentir la convergence du modèle global, dégrader sa précision et introduire des biais.

Un problème central dans ce contexte est celui de la sélection des clients. À chaque tour de communication, le serveur doit choisir quels clients participeront à l'entraînement ou à l'évaluation. Dans les approches classiques, cette sélection est souvent considérée comme un problème d'échantillonnage, de classement ou de filtrage : il s'agit de choisir un sous-ensemble de clients selon des critères tels que la disponibilité, l'utilité statistique, la rapidité estimée ou la représentativité des données. Néanmoins, cette vision reste limitée, car elle ne contrôle pas explicitement la quantité de travail attribuée à chaque client. Deux clients sélectionnés peuvent recevoir des charges de travail identiques alors qu'ils possèdent des capacités matérielles, énergétiques et statistiques très différentes. Inversement, un client intéressant du point de vue de ses données peut être lent ou peu fiable, tandis qu'un client très rapide peut posséder une distribution de données peu représentative.

Dans les systèmes synchrones, cette difficulté est amplifiée par le phénomène des *stragglers*. Le serveur doit attendre la fin du travail local des clients sélectionnés avant de pouvoir agréger leurs mises à jour. Par conséquent, la durée d'un tour est souvent déterminée par le client sélectionné le plus lent. Une mauvaise décision de sélection peut donc augmenter fortement le temps total d'entraînement, même si la majorité des clients sont rapides. De plus, dans des environnements de périphérie ou mobiles, la consommation énergétique devient une contrainte essentielle. Un algorithme qui sélectionne systématiquement les mêmes clients rapides ou énergétiquement efficaces peut réduire le temps d'exécution à court terme, mais il risque aussi de concentrer l'effort sur un petit groupe de participants, d'épuiser certains appareils, de diminuer l'équité de participation et de réduire la diversité des données utilisées par le modèle global.

Cette thèse part du constat que la sélection des clients ne doit pas être abordée uniquement comme le choix binaire d'un sous-ensemble de participants. Elle peut être formulée comme un problème d'ordonnancement, dans lequel les clients sont vus comme des ressources hétérogènes et la charge de travail d'un tour d'apprentissage fédéré est divisée en unités plus fines appelées tâches. Une tâche correspond à une portion de données locales, par exemple un sous-ensemble d'images ou d'exemples textuels appartenant à une ou plusieurs classes. Le serveur ne décide donc pas seulement quels clients participent, mais aussi quelle quantité de données chaque client doit traiter. Cette formulation permet d'adapter la charge de travail aux capacités et à la fiabilité des clients, tout en tenant compte de critères liés au temps, à l'énergie, à la qualité du modèle, à la diversité des données et à l'équité de participation.

L'objectif général de cette thèse est ainsi de proposer des méthodes de sélection et d'allocation de charge pour l'apprentissage fédéré synchrone en environnement *cross-device*. Le travail vise à améliorer simultanément l'efficacité du système, la progression de l'apprentissage, la robustesse face à l'hétérogénéité et l'équité entre participants. Plus précisément, la thèse étudie comment des méthodes issues de l'ordonnancement, de la programmation dynamique et des métaheuristiques peuvent être utilisées pour contrôler finement la participation des clients et la quantité de données traitées localement. Elle étudie également comment cette approche peut être étendue à une formulation multi-objectifs intégrant non seulement le temps et l'énergie, mais aussi des critères propres à l'apprentissage fédéré, tels que la qualité de la distribution des classes, l'utilité historique des clients, la diversité de participation et la fiabilité des appareils.

Démarche adoptée. La démarche adoptée repose d'abord sur une modélisation du problème de sélection des clients comme un problème d'ordonnancement de tâches. Dans cette formulation, un tour d'apprentissage ou d'évaluation fédérée est représenté par un ensemble de tâches à attribuer aux clients disponibles. Chaque client est caractérisé par des informations de profilage, telles que son temps d'exécution estimé, sa consommation énergétique estimée, sa disponibilité et son historique de participation. Cette représentation permet de dépasser la sélection binaire traditionnelle et d'introduire un contrôle plus précis de la charge de travail. Un client rapide et fiable peut recevoir davantage de tâches, tandis qu'un client plus lent, moins fiable ou plus coûteux en énergie peut recevoir une charge réduite, voire ne pas être sélectionné selon les objectifs du tour.

La première contribution méthodologique de la thèse consiste à proposer deux algorithmes optimaux d'ordonnancement orientés temps et énergie, appelés *MEC* et *ECMTC*. Ces deux algorithmes sont conçus pour le contexte de l'apprentissage fédéré synchrone et exploitent une structure de programmation dynamique. *MEC* vise d'abord à minimiser le *makespan*, c'est-à-dire la durée maximale d'exécution parmi les clients sélectionnés, puis à réduire la consommation d'énergie lorsque plusieurs solutions présentent le même temps d'exécution. Cet algorithme répond au problème des *stragglers* en cherchant à répartir la charge de manière à réduire la durée du tour. *ECMTC*, au contraire, donne la priorité à la minimisation de la consommation énergétique sous une contrainte de temps. Il cherche donc une allocation qui respecte une limite temporelle tout en réduisant l'énergie consommée. Ces deux algorithmes permettent d'étudier formellement le compromis entre temps et énergie et de montrer que la sélection de clients peut être traitée comme un problème d'allocation optimale de charge.

La seconde contribution, plus générale, est le développement du *framework MetaCS-FL*. Ce *framework* étend la perspective d'ordonnancement au-delà des seuls objectifs de temps et d'énergie. Alors que *MEC* et *ECMTC* se concentrent principalement sur les métriques système, *MetaCS-FL* formule la sélection des clients comme un problème multi-objectifs intégrant plusieurs dimensions complémentaires : la durée d'exécution du tour, la consommation énergétique totale, l'utilité statistique des clients, la qualité de la distribution des classes, la diversité de participation et la fiabilité. Le *framework* est conçu pour être flexible et piloté par événements. Il peut réutiliser une solution précédente lorsque le contexte reste stable, ou déclencher une nouvelle sélection lorsque des changements importants apparaissent, par exemple dans la disponibilité des clients, la progression de l'entraînement ou la composition du groupe de participants.

Dans l'implémentation étudiée dans cette thèse, *MetaCS-FL* utilise la solution produite par *ECMTC* comme point de départ, puis l'améliore au moyen d'une métaheuristique de type *Large Neighborhood Search* (LNS). Cette stratégie permet de combiner les avantages d'une solution initiale structurée avec la flexibilité d'une recherche métaheuristique capable d'incorporer plusieurs objectifs. La fonction de coût utilisée par *MetaCS-FL* agrège différents critères pondérés. Le *makespan* mesure la durée du tour, l'énergie mesure le coût énergétique de la charge assignée, la diversité encourage la participation de clients variés au fil des tours, et les critères liés aux classes favorisent des sélections plus représentatives des distributions de données. L'utilité historique, notamment fondée sur la perte observée lors des entraînements précédents, permet de donner plus d'importance aux clients dont les mises à jour sont susceptibles de contribuer au progrès du modèle. La fiabilité, quant à elle, est estimée à partir de l'historique de disponibilité et d'achèvement des tâches. Elle permet de limiter la quantité de travail assignée à des clients instables sans les exclure totalement.

Un aspect important de la démarche concerne la confidentialité des informations relatives aux données locales. Pour sélectionner des clients en tenant compte de la distribution des classes, le serveur pourrait être tenté de demander des informations détaillées sur les données de chaque client. Or, ces informations peuvent révéler des propriétés sensibles. La thèse intègre donc un mécanisme de divulgation préservant la confidentialité, fondé sur la confidentialité différentielle. Au lieu de transmettre la distribution exacte de leurs classes locales, les clients peuvent communiquer des statistiques bruitées. Le serveur utilise alors ces informations approximatives pour guider la sélection tout en réduisant le risque de divulgation directe des distributions locales. Cette composante permet d'étudier le compromis entre qualité de la sélection, confidentialité et performance globale.

La démarche expérimentale vise à évaluer les méthodes proposées dans des scénarios réalistes et variés. Les expériences sont menées sur des tâches d'apprentissage supervisé utilisant des jeux de données publics : CIFAR-10 pour la classification d'images, Fashion-MNIST pour la reconnaissance d'articles vestimentaires, et Emotion pour la classification de textes émotionnels. Les données sont réparties entre les clients selon des configurations IID et non-IID afin d'étudier l'effet de l'hétérogénéité statistique. Les expériences sont déployées sur des nœuds Grid'5000 avec des dispositifs émulés, ce qui permet de contrôler les profils de calcul, de communication et de consommation énergétique. Les méthodes proposées sont comparées à plusieurs algorithmes représentatifs de la littérature, notamment FedAvg comme référence aléatoire, *MEC* et *ECMTC* comme ordonnanceurs optimaux orientés système, ainsi que des méthodes guidées par l'utilité, la diversité, la réputation, la disponibilité ou l'équité, telles que Oort, DivFL, ECSM, FedCAB et RIFLES.

L'évaluation couvre plusieurs dimensions. Une première série d'expériences étudie la disponibilité statique, où tous les clients restent éligibles pendant l'entraînement. Elle permet d'analyser la participation des clients, la distribution des charges de travail, la précision du modèle, le nombre de tours nécessaires pour atteindre une précision cible, le temps total d'entraînement, la consommation énergétique totale et l'équité. Une deuxième série étudie l'effet de la confidentialité différentielle sur les statistiques de classes utilisées par *MetaCS-FL*. Une troisième série analyse la scalabilité du *framework* lorsque le nombre de tâches ou de clients augmente, ainsi que le surcoût de calcul côté serveur. Une quatrième série évalue la sensibilité aux paramètres du *framework*, notamment les poids de la fonction objectif, les déclencheurs de resélection, la réutilisation des solutions et les paramètres de la métaheuristique. Enfin, des scénarios dynamiques sont étudiés, avec des clients rejoignant tardivement le système ou devenant intermittents, afin de mesurer la robustesse de *MetaCS-FL* lorsque l'ensemble des clients disponibles évolue pendant l'apprentissage.

Résultats obtenus. Les résultats montrent d'abord que la formulation par ordonnancement est pertinente pour l'apprentissage fédéré synchrone. Les algorithmes *MEC* et *ECMTC* démontrent que le contrôle de la quantité de travail assignée à chaque client permet d'améliorer les compromis entre temps d'exécution et énergie. *MEC*, en minimisant d'abord le *makespan*, réduit l'impact des clients lents et tend à raccourcir la durée des tours. *ECMTC*, en minimisant d'abord l'énergie sous contrainte temporelle, favorise des allocations plus économes, souvent concentrées sur des clients plus efficaces. Ces deux approches montrent que l'optimisation système peut être formulée et résolue de manière structurée. Elles révèlent également leurs limites : optimiser principalement le temps ou principalement l'énergie ne suffit pas toujours à produire les meilleurs modèles, surtout lorsque les données sont non-IID et que la représentativité statistique des clients joue un rôle essentiel.

Les expériences préliminaires associées à *MEC* et *ECMTC* confirment ce compromis. Dans les configurations étudiées, *MEC* réduit fortement le temps total d'entraînement par rapport à une sélection aléatoire, tandis que *ECMTC* réduit la consommation énergétique sous contrainte de délai. Par exemple, dans les expériences d'ordonnancement avec une charge de travail représentant 25% des données d'entraînement et de test, *MEC* réduit le temps total d'entraînement de 54,33% par rapport à la stratégie aléatoire, alors que *ECMTC* réduit la consommation énergétique totale de 26,57%. Lorsque la charge de travail augmente à 50%, *MEC* réduit le temps total d'entraînement de 67,52%, tandis que *ECMTC* conserve un avantage énergétique. Ces résultats montrent que les deux algorithmes atteignent les objectifs pour lesquels ils ont été conçus, mais aussi que le choix d'un objectif principal influence fortement la nature des clients sélectionnés et la distribution de la charge.

Les résultats montrent ensuite que *MetaCS-FL* offre un compromis plus équilibré que les méthodes centrées sur un seul critère. Sous disponibilité statique, le *framework* obtient les meilleurs résultats combinés dans les expériences à 100 clients. Les améliorations les plus importantes apparaissent notamment dans le cas CIFAR-10 non-IID, où *MetaCS-FL* réduit le temps total d'entraînement jusqu'à 88,85% et la consommation énergétique totale jusqu'à 84,45% par rapport à FedAvg. Dans ce même contexte, le nombre moyen de tours nécessaires pour atteindre la précision cible passe d'environ 92 tours avec FedAvg à environ 30 tours avec *MetaCS-FL*. Ces résultats indiquent que le *framework* ne se contente pas d'accélérer les tours individuellement : il favorise également des sélections qui améliorent la progression de l'apprentissage.

La comparaison avec *MEC* et *ECMTC* met en évidence l'intérêt de l'approche multi-objectifs. *MEC* peut réduire efficacement le temps d'un tour en répartissant la charge sur plusieurs clients, mais cette stratégie peut augmenter l'énergie consommée, le coût de sélection ou sélectionner des clients dont les données ne contribuent pas suffisamment à la convergence. *ECMTC* peut réduire l'énergie en privilégiant des clients efficaces, mais cette logique peut concentrer la participation sur un petit nombre d'appareils, diminuer l'équité et biaiser le modèle lorsque les données sont hétérogènes. *MetaCS-FL* dépasse ces limites en combinant l'efficacité système avec des critères d'apprentissage et de participation. Il peut donc sélectionner des clients qui ne sont pas seulement rapides ou économes, mais aussi utiles, représentatifs, diversifiés et fiables.

Les résultats obtenus face aux méthodes de la littérature montrent également l'intérêt de cette intégration. Les méthodes guidées par l'utilité peuvent accélérer l'apprentissage dans certains cas, mais elles ne contrôlent pas nécessairement l'énergie, l'équité ou la charge de travail. Les méthodes fondées sur la diversité améliorent la couverture statistique, mais elles peuvent sélectionner des clients coûteux ou lents. Les approches fondées sur la réputation ou la disponibilité peuvent être efficaces lorsque l'historique est stable, mais elles sont moins robustes lorsque de nouveaux clients apparaissent ou lorsque la disponibilité varie fortement. *MetaCS-FL* se distingue parce qu'il ne dépend pas d'un seul signal. Il intègre simultanément les dimensions système, statistiques et dynamiques dans une décision d'ordonnancement, ce qui explique sa stabilité sur plusieurs jeux de données, distributions et scénarios.

L'analyse de la confidentialité différentielle montre que l'utilisation de statistiques de classes bruitées reste exploitable pour guider la sélection. Les performances de *MetaCS-FL* avec divulgation différentiellement privée restent proches de celles obtenues avec les distributions non bruitées. Dans les expériences rapportées, cette variante réduit même légèrement le temps d'entraînement de 0,65% et améliore l'équité de 8,88%, bien qu'elle puisse augmenter la consommation d'énergie et le temps de sélection. Ce résultat montre que les informations de distribution n'ont pas besoin d'être exactes pour être utiles : des signaux approximatifs peuvent suffire à orienter la sélection vers des groupes de clients plus représentatifs, tout en limitant l'exposition directe des données locales.

Les analyses de scalabilité et de surcoût côté serveur indiquent que *MetaCS-FL* reste praticable malgré sa formulation plus riche. Son surcoût est supérieur à celui de méthodes légères comme FedAvg, Oort ou ECSM, ce qui est attendu puisque le *framework* résout un problème plus complexe. Cependant, il reste beaucoup plus scalable que les approches exactes *MEC* et *ECMTC* lorsque le nombre de tâches et de clients augmente. Cette propriété vient notamment de l'utilisation d'une métaheuristique à temps borné, de la réutilisation de solutions précédentes et de déclencheurs de resélection évitant de recalculer inutilement un nouvel ordonnancement à chaque tour. Le *framework* trouve ainsi un compromis entre qualité de décision et coût de calcul côté serveur.

Les scénarios dynamiques confirment la robustesse du *framework*. Dans les expériences avec clients rejoignant tardivement le système, *MetaCS-FL* réduit le temps total d'entraînement jusqu'à 82,09% et l'énergie totale jusqu'à 76,79% par rapport à FedAvg. Lorsque des clients performants arrivent tardivement, le *framework* peut les exploiter en leur attribuant une charge utile. Lorsque les clients arrivant tardivement sont moins performants, il limite leur charge sans les exclure systématiquement. Cette capacité d'adaptation est importante pour les environnements réels, où les clients ne sont pas tous présents dès le début de l'entraînement.

Dans les scénarios avec disponibilité intermittente, *MetaCS-FL* obtient également les meilleurs résultats globaux. Il réduit le temps total d'entraînement jusqu'à 79,58% et l'énergie totale jusqu'à 62,84% par rapport

à FedAvg. Ces gains ne proviennent pas simplement de la sélection d'un plus petit nombre de clients. Ils résultent de l'utilisation de scores de fiabilité pour contrôler la charge admissible des clients instables. Les clients moins fiables peuvent rester éligibles, mais leur charge est réduite afin de limiter le risque d'échec ou de travail non terminé. Cette stratégie préserve une certaine diversité de participation tout en améliorant la robustesse opérationnelle.

Conclusions. Cette thèse montre que la sélection des clients en apprentissage fédéré peut être comprise comme un problème d'ordonnancement multi-objectifs plutôt que comme un simple échantillonnage de participants. La division du travail en tâches et l'allocation fine de ces tâches aux clients permettent de mieux exploiter les capacités hétérogènes des appareils tout en tenant compte de la qualité statistique des données. Les algorithmes *MEC* et *ECMTC* établissent les bases de cette approche pour l'optimisation du temps et de l'énergie. *MetaCS-FL* généralise ensuite cette perspective en intégrant des critères d'efficacité, d'utilité, de diversité, de confidentialité et de fiabilité. Les résultats expérimentaux montrent que cette approche améliore simultanément le temps d'entraînement, la consommation énergétique, la convergence et la robustesse dans des scénarios statiques et dynamiques.

Les limites du travail ouvrent plusieurs perspectives. La thèse se concentre sur un *framework* synchrone et centralisé, avec une agrégation de type FedAvg, des paramètres locaux fixes et des appareils émulés. Les poids de la fonction objectif sont définis à l'avance, ce qui peut limiter l'adaptation aux différents stades de l'entraînement ou aux contraintes applicatives. Des extensions futures pourraient adapter *MetaCS-FL* à l'apprentissage fédéré asynchrone ou semi-asynchrone, introduire des pondérations dynamiques des objectifs, optimiser conjointement la sélection, la charge de travail et les hyperparamètres locaux, ou valider le *framework* sur des dispositifs réels. D'autres directions incluent la prise en compte de réseaux de communication plus réalistes, de clients malveillants, de contraintes temporelles strictes ou d'applications critiques. Malgré ces limites, les contributions de cette thèse montrent que l'ordonnancement multi-objectifs constitue une abstraction prometteuse pour concevoir des systèmes d'apprentissage fédéré plus efficaces, plus équitables et plus robustes.

Genitori genitoque
laus et iubilatio,
salus, honor, virtus quoque
sit et benedictio,
procedenti ab utroque
compar sit laudatio.

"To the Begetter and the Begotten
be praise and jubilation;
salvation, honor, power,
and blessing;
and to the One proceeding from both
be equal praise."

— S. Thomae de Aquino, *Pange lingua*, in *Officium corporis Christi*

Acknowledgments

Castígans castigávit me Dóminus: et morti non trádidit me.
(*Psalmus* CXVII, *Confitemini Domino*)

For over a decade, I have intoned this psalm at Lauds in the *Divinum Officium*; yet only in recent years have I come to know its true meaning for me. Yes, Lord, Thou hast severely tried me, and I have truly had to wage war against the world, the flesh, and the devil. Yet Thou wast with me: guiding me, carrying me, and blessing me, even through the depths of night and the deserts of loneliness. And therefore, whatsoever good I have done, and whatsoever good I may yet do, I ascribe to Thee alone. For Thou didst teach me once, and it left a deep mark upon my soul: ***quia sine me nihil potestis facere***, "for without Me you can do nothing" (*Ioannes* XV, V). With all my heart, I thank Thee, O loving Savior, for all the graces Thou hast given me. I have been unfaithful, negligent, and a transgressor; yet I do not wish to be ungrateful. I wish to show Thee my gratitude by giving myself wholly and entirely to Thee. I know that this life is but a passing breath; soon the time of mercy shall draw to its close, and judgment shall be set. I also know that I shall be tried again, perhaps even more severely, in the moments yet allotted to me. For this, I beseech Thee, forsake me not, but remain with me unto the end, that I may be counted among Thine elect.

Life is a gift from God, Who asks us to be His collaborators. For this reason, I give immense thanks to my father and mother, Renato and Neide, for bringing me into this life, and for raising me with such love, patience, toil, tears, and sacrifice. You gave me every foundation I could have asked for in order to become a person of character, faith, and perseverance. Whatever good there is in me bears, in some way, the imprint of your generosity and love. I owe, in a special way, to my father my first contact with computing: it was he who brought into our home our first personal computer, at a time when none of us knew exactly how to use it, and when the Internet still came to us slowly, through a dial-up connection, with little to explore by today's standards. Though it worked only for a brief time, that machine awakened in me a curiosity that would not leave me; I remember experimenting with it, trying things I did not understand, and eventually rendering the operating system inoperable. In that small beginning, half wonder, half mischief, there was already the seed of a path that would later become my own. And when unemployment knocked at our door, you had to draw upon admirable creativity and talent in woodworking in order to stem the great financial difficulty we had to face. We lost much, including our home, furniture, and possessions, but, thanks be to God, they were only material things. Thus, early in life, I learned a hard but precious lesson: that the things we lose are not always the things that matter most. Today, perhaps, none of those things would still have any value. But you are still here, and that, to me, has been the greatest gift. And I dedicate this even more especially to my mother, who always believed in my potential. I myself am a witness to all that she had to suffer in order not to give up, bleeding, quite literally, for us, as she worked at a humble job that barely covered our modest school tuition. And now that we have passed together through so many sufferings, anxieties, uncertainties, hunger, and want, I thank God all the more for your presence, your strength, and your fidelity. May He repay you abundantly for all that you have done for me, and may He grant us the grace to remain united, whatever may yet come. Who could have imagined that I would come this far after all that we have endured? My diploma will bear my name; but if I could, I would exchange it for hers.

A friend for life. If I had to describe what my brother means to me, that would be one of the truest ways to say it. For much of my life, he was there by my side. We grew up together, made friends together from an early age, played countless games together, and, of course, had our share of fights along the way. Ultimate Mortal Kombat III, in particular, would have much to say about that. The truth is that he has always been there for me. For me, friendship has another name: Renan. I trusted his judgment so much, and was so convinced that he knew the right path to take, that I ended up following nearly the same academic path, from school to university. Brother, know this: you have been my closest inspiration for a very long time. You are brilliant, hardworking, and deserving of every happiness in life. I wish I could have helped you more than I have. Still, life is hard, and not all our plans unfold as we expected. May you always have faith in God, and may He reward you greatly. At a decisive moment for this thesis, when everything seemed to be

falling apart, you were there once again. Once more, you extended your friendly hand and came to my aid in a way that I will carry with gratitude for the rest of my life. More than once, our parents told us that we must remain united, because, in the end, one's truest friends are found within one's own family. And I sincerely have to agree with them. I believe that, as time passes and life brings its many turns, it may one day be you and me again: no longer young, but old and frail. To be honest, I would not mind that at all.

"Somewhere, in some unknown place, when nothing else still mattered, I met you. That afternoon, when I found you, the world seemed to hold only you and me. It could not have been better! Lives that meet. Your eyes tell me, without needing to speak [...] Lives that meet, without any explanation; paths leading to the same place." These verses are deeply meaningful to me, for they describe with great precision the moment I met you, Thamyres, love of my life. Where could I possibly begin? We were so young, and yet I already knew that you were exactly the one I needed by my side. It has been an incredible journey of love, discovery, grace, forgiveness, sadness, joy, and companionship. We have been through so much together that it is hard to say where our story truly began; all I know is that today, there is no me without you. To be honest, what did we have to offer each other besides love? Nothing. We grew up poor and amid so many uncertainties, but still, we did not care much about any of that. You know it well: not every day was easy. We had to face many difficult moments and make many tough decisions. When I lived in Bordeaux, the days were simply not the same without your company. And I found myself repeating the verses of Alceu Valença, a great Brazilian musician, about loneliness: *"Loneliness is fierce, a friend of the hours, time's first cousin, and it makes our clocks move slowly, causing my heart to fall out of rhythm; the loneliness of the stars, the loneliness of the moon, the loneliness of the night, the loneliness of the street."* And I meant every word. For many days, that was how life felt to me. But the most beautiful part of it all is that we went through those adversities together. Sometimes, it is not only about the destination, but about the path we take to reach it. And if that is true of us, then I can say that I have already been blessed beyond measure. So, as Presley once sang so beautifully: *"Take my hand. Take my whole life too. For I can't help falling in love with you."*

I cannot help but remember, with tenderness, my cat, *Valente*, who, from the very first day and throughout the past twelve years, was deeply loved and cared for by us. From my undergraduate studies to the completion of my thesis, he accompanied me throughout my academic journey. He kept us company through countless days and nights and remained by our side until his final day, even when illness weighed heavily upon him. He was brave until the end and fought hard for three months just to stay with us a little longer, bringing us happiness and love. Valente, you will never be forgotten, for you made our lives special for such a long time. Thank you for choosing us and for trusting us as your family. To my grandparents, uncles, aunts, cousins, and relatives, I give thanks for the affection, encouragement, prayers, memories, and moments of companionship that have marked my life. All of you have a special place in my heart, for each of you, in your own way, has helped shape the person I am. In you, I recognize something of the history from which I come, and for that I am sincerely grateful. In particular, I would like to thank my uncle Ari and my aunt Nanci, and my cousins Filippe and Gabrielli, for the many joyful moments we shared together in their blessed home. In the same way, I would like to thank my uncle Gerôncio and my aunt Nilce, and my cousins Thiago and Walter, for everything they did for us, especially during so many difficult times for my family and me. They welcomed us with open arms week after week, and I am certain that such charity will not go unnoticed by God. For me, it felt like a second home. Thiago and Walter, in particular, you have taught me so many things about computing, even when you did not realize it. In fact, much of what I know I learned simply by watching the two of you work on your shared computer. My early interest in computers reached another level after all the time I spent with you both. Thank you for lending me your computer, your printer, and whatever else I needed on so many occasions. I suppose I never had the chance to say how grateful I am for all of this. And I am truly sorry that I was far away when your beloved father passed away. I wished that I could have been there with you, but circumstances did not allow it. Still, God granted me the grace of seeing him once more in my dreams, and there I had the chance to tell him that I loved him and to entrust him to God's mercy. May we always remember what Our Lord has promised us: ***Ego sum resurrectio et vita. Qui credit in me, etsi mortuus fuerit, vivet.***, "I am the resurrection and the life. He who believes in me, though he may die, he shall live" (*Ioannes* XI, XXV).

To my dear advisors, I offer my sincere gratitude. During my undergraduate studies, I met Lúcia; during my master's degree, Cristina; and during my doctorate, Laércio. Along this path, I had the privilege of learning from distinct minds, perspectives, and ways of thinking. Working with each of you was a genuine pleasure, and I am grateful not only for your guidance, but also for the valuable feedback that challenged me, opened new ways, and gave me many opportunities to grow as a researcher and as a person. You gave me opportunities that marked my life deeply. For instance, my first flight by plane, to present at a conference in another state of Brazil, took place at the very beginning of my PhD. Less than five months later, I was traveling abroad for the first time, where I had the opportunity to meet Laércio in person and to define the very basis of my doctoral research. During all these years, but especially throughout my PhD project, I developed many skills that go far beyond methodology: critical thinking, scientific maturity, intellectual discipline, clarity in writing, confidence in presenting my ideas, and the ability to transform feedback into opportunities for growth. The journey was intense from beginning to end, for many reasons, including the demanding administrative procedures involved in being enrolled in two universities, scientific challenges, deadlines, uncertainties, distance, and the personal demands that accompanied the whole project. I also faced failures along the way, including rejected works that, in some sense, eventually led to other impactful opportunities and helped me understand that even setbacks can become part of a greater path. To each of you, Lúcia, Cristina, and Laércio, I offer a direct and sincere thank you. I tried, within my possibilities, to give the very best I could in order to bring this work to a successful conclusion. Not everything went as planned, and I know that I did not reach all the contributions I once hoped I would. Some circumstances set me back, and some obstacles were beyond what I could fully overcome. Even so, I hope that this thesis may still bear witness to my effort, my gratitude, and the respect I have for the trust you placed in me. May God bless each of you abundantly and repay, in ways I never could, all the good you have done for me.

My heartfelt gratitude also goes to Luan Teylo and Isaac Balster, two friends whom I had the joy of meeting in Bordeaux, and who helped me in so many practical and generous ways, making those first difficult steps in a new country much easier. I must say that, more than once, Luan literally left me the keys to his home and welcomed me warmly, as if I were an old friend or a member of his own family. Your attitude filled me with deep admiration, my friend. Few people outside my own family have cared for me with such generosity. During the time I lived in Bordeaux, I was able to know you better and to appreciate your singular personality. Curiously, we still happen to meet here and there at events, such as in Nantes, France, and in Atlanta, in the United States. Even on the few occasions when we meet, I am always reminded of the generosity and good humor with which you welcomed me in those first days abroad. Isaac, I only wish you could have stayed longer in Bordeaux. Those few days when you were still there bring me many good memories. It was my first Christmas and New Year's Eve away from my loved ones, but you were there. And many things were much easier with your company. That water bottle you gave me? It has now been with me for more than two and a half years of daily use. Many things could have gone terribly wrong for me, had it not been for your solidarity and friendship. I hope that one day I may be able to repay, at least in part, what you both have done for me. I am certain that every good thing you have done for me has been seen by God, and I sincerely thank you both and wish you every blessing and happiness in life.

I would like to thank the friends I had the great pleasure of meeting during my teenage years, especially Richard Azevedo, Gabriel Rabello, and Juan Mendes. Under different circumstances, but with the same sincere affection, each of you became part of my life in a way that I still remember with gratitude. You made my days easier when I was facing one of the most difficult periods of my life, one that, at times, still haunts me to this day. Learning to smile again was what I needed most, and you all gave me much more than that. Richard, all those countless hours of chatting and gaming are priceless to me. Could we have done things that were more useful than that? Certainly, we could have. But perhaps that was exactly what I needed at that time. And when you told me about the course you were taking, it inspired me to seek something for myself as well, which eventually led me to technical high school. Gabriel, you in particular walked beside me for a longer season, especially during our internship. There were so many funny and pleasant moments, but also some truly sad ones, and you know well what I mean. I wish I could have done more for you at that time, but I did not know how. Years later, when I had to choose my Confirmation sponsor, deciding on you

felt quick and natural. Seeing the loving father you have become fills me with joy. Juan, my dear musician friend, you followed a path that had somehow also been present in me since childhood, from the days of my first cheap flute. Yet you were the one who took the great step, moving forward with courage. And I miss our music sessions so much, whether at my home or yours, or perhaps our funny live performances on the buses on the way home from school. Those were the days: simple and happy days, my friend. I also deeply admire the beautiful family you have built. I know that life has led us along different paths and that it has become difficult to find the time to give the three of you the attention and presence I wish I could offer. But know this: whatever happens in life, I will always carry with me the happy moments we shared when we were younger. You taught me how to face difficult moments with a lighter heart, and I am grateful for that. This made all the difference during my academic journey. May God bless your lives with peace, joy, and every grace you need.

To all my brothers and sisters in Christ whom I have had the grace of meeting throughout these many years of service to the Church, I offer my heartfelt gratitude. You have inspired me in so many ways and taught me how to face the worst adversities with the humble certainty that God knows better than we do, and that His providence never fails. In particular, I would like to thank those who were preparing for Confirmation at the time, for placing their trust in me in such a serious task as teaching them, with due diligence, the truths of the Gospel and of the Catholic faith. Of all my duties, this was undoubtedly the most rewarding. In a sense, it also helped me along my academic path, as it required me to research, read, prepare, explain, and communicate complex matters with clarity and responsibility. I offer a special thank you to my Confirmation godchildren, Rosa, Reynaldo, and Gabrielli, for seeing in me someone worthy of accompanying them in such a serious and beautiful step of their Christian lives. I would also like to thank Fathers Carlos Alberto Nascimento and Daniel Mourão, from Paróquia Nossa Senhora da Saúde, in Curicica, who, from my earliest years, taught me about God's love with patience, dedication, and pastoral charity. They have labored tirelessly in this mission, and they continue to do so to this day. I thank Father Alex Muniz, also from that parish and a friend since childhood, who generously celebrated a special Mass for me. I would like to offer particular thanks to Father Jean-Marie Mavel, of Chapelle Notre-Dame du Bon Conseil, in Bordeaux, whose priestly dedication, example, and fidelity to the service of God deeply marked me.

During nearly ten years of formal and informal collaboration with PCERJ, the Civil Police of the State of Rio de Janeiro, I made many friends who marked an important chapter in my life. There I met true heroes, whose lives inspired mine and taught me, by example, the meaning of courage, duty, and sacrifice. In particular, I thank Octaviano Dias, one of the most wonderful people I could ever hope to meet. With him, many ideas came to fruition, and with creativity, we accomplished numerous tasks and overcame many challenges together. He helped me during times when the State itself gravely failed me, and his support greatly helped me carry on through a time of uncertainty. His brilliant mind, expressed through intriguing daily questions and thoughts, pushed me further than I would have gone on my own, which directly contributed to my academic growth. I also thank Eladio Casal, from CTF. In one of our pleasant conversations, he encouraged me to pursue a bachelor's degree in Computer Science. I no longer remember the exact day, but that conversation truly motivated me to look toward the future with greater purpose, which eventually led me to enroll at UFF. To Octaviano, Eladio, and many others who come to my mind, I am truly grateful for everything. May God reward you abundantly for all the good you have done for me.

During my experience as a scholarship student at STI, the Office of Information Technology at UFF, I had the opportunity to learn, contribute, and grow. There, I had the opportunity to meet colleagues, particularly my former manager Marcos Côrtes and my former teammates Rodrigo Zacarias and Flávio Cruz. Our daily meetings helped me broaden my ideas, and that environment offered many opportunities to learn new tools, some of which I still use to this day and which helped me carry out academic activities. I wish you all success, joy, and every good thing in your lives.

My gratitude also extends to the many teachers who guided my first steps in learning, from childhood through high school, and to the professors who accompanied me along my academic journey. Though I cannot name each of you individually, if the reader is a professor and has ever met me, then you are part of

this gratitude, and I thank you for the effort in such a beautiful vocation. At the technical high school Escola Técnica Estadual República, in Quintino Bocaiúva, I learned the foundations of computing. At Universidade Federal Fluminense, in Niterói, and at Université de Bordeaux, in Bordeaux, I was introduced to, studied, and mastered many of the scientific and technical principles that made this work possible. To all of you, I offer my gratitude and admiration, and may God bless your mission.

During my stay at Inria Bordeaux, I also had the opportunity to meet outstanding researchers and professionals, most of whom were from the STORM, TOPAL, and TADaaM teams. I met people of different nationalities, and the experience was deeply enriching, both professionally and personally. There, I had the chance to present my research progress, discuss research topics, attend meetings and presentations, and, overall, broaden my understanding of how collaborative research is carried out in an international environment. Once again, the list of thanks is long, but I would like to express my gratitude to two people in particular. Mihail Popov, from the very first day there, when I was not even enrolled at the university, you welcomed me with kindness, helped me find my place, and made that new environment feel less intimidating. You are generous of heart, and you never hesitated to include me in conversations, even when people were speaking French and I could not follow. I admire you deeply, and you are truly one of a kind. Ellie Corrêa da Costa, amid so many circumstances that seemed to happen to me with great frequency, such as problems with trains, tickets, accommodation, and other matters, you never hesitated to help me, with remarkable competence and efficiency. And yes, someone might say that it was merely your job; but to me, it was much more than that. At times, it made the difference between success and failure, between being lost and finding a way forward, between feeling helpless and knowing that someone was there to help. At times, it almost seemed as though you were competing with my guardian angel, so complicated had some things become for me. I am deeply grateful to both of you and to many others there for making my stay fruitful, safe, and remarkable. May God provide for you all, reward your generosity, and bless your paths abundantly.

I am also grateful to the many current and former fellow students I met at both universities, whether in the HPC+Cloud laboratory (UFF), in STORM's open space (UB), or beyond, for their companionship and the conversations and exchanges that enriched my academic journey. The list is long, but I would like to thank, in particular, from UFF: Rafaela Brum, Ronald Campbell, Camila Lopes, Mateus de Melo, Artur Bittencourt, Daniel Sodré, Miguel de Lima, Bernardo Gallo, Luciano Andrade, Alana Gadelha, Matheus Marotti, and Lucas Serrano; and from UB: Albert D'aviau De Piolant, Nicolas Ducarton, Thomas Morin, Alice Lasserre, Vincent Alba, Jean-François David, Diane Orhan, Lana Scravaglieri, Radjasouria Vinayagame, and Maxime Gonthier. I wish you all success, joy, and many blessings from God on the paths ahead.

My gratitude also goes to the many researchers and professionals with whom I have collaborated, and in particular those whose guidance, generosity, and work have contributed to my growth: Luan Teylo, Alba Melo, Natalia Martins, Daniel de Oliveira, Claude Tadonki, Felipe Portella, Paulo Estrela, Renzo Malini, Bruno Lopes, José Viterbo, Vinod Rebello, Lucas da Costa, Artur Pessoa, and many others. To all of you, I offer my sincere thanks for the collaboration and opportunities that have helped shape my academic path.

I offer special thanks to Pierre Ramet and Denis Trystram for serving on my *Comité de Suivi Individuel* (individual monitoring committee), as required by UB, and to Allan de Souza, Flávia Delicato, and Valmir Barbosa for serving on my thesis proposal defense committee, as required by UFF. I would also like to thank Shadi Ibrahim, Alfredo Goldman, Valmir Barbosa, César De Rose, Débora Saade, and Flávia Delicato for serving on my thesis defense jury. Thank you for your time and effort in helping me improve my research. You all played an important role in this process, and I am sincerely grateful for your meaningful feedback.

I gratefully acknowledge CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), CAPES-PrInt (Programa Institucional de Internacionalização), Inria (Institut national de recherche en sciences et technologies du numérique), and Petrobras (Petróleo Brasileiro S.A.) for the financial support granted throughout my academic journey. I would also like to thank Olivier Beaumont, Laércio Pilla, Francieli Boito, Lúcia Drummond, Cristina Boeres, Olivier Aumage, Alexandre Plastino, and many others who helped me, in different ways, to obtain the stability and opportunities I needed for travel, event participation, courses, and other valuable experiences that allowed me to pursue my goals with peace of mind. I am also grateful to

the DECoHPC joint team for making possible additional research visits, collaborations, and exchanges that contributed significantly to my academic development and broadened my international experience.

I am also grateful to the Grid'5000 team and its maintainers, since the experiments carried out in this thesis would not have been possible without the resources they provide.

Dear reader, have you noticed how long these acknowledgments have become? Yet it is probably not even ten percent of all the people I would like to thank in particular. And to you, who have read it up to this point, I must include you among them as well, for it seems that, in some way, you have chosen to accompany me through these words and through a part of my story. What I have truly sought to do here is to follow the teaching of St. Thomas Aquinas: ***Iustitia est habitus secundum quem aliquis constanti et perpetua voluntate ius suum unicuique tribuit.***, "Justice is the habit by which someone, with a constant and perpetual will, gives to each person their due." (Summa Theologiæ, II$^{\underline{a}}$-IIæ, q. 58, a. 1). So, I suppose I have managed to do a small part of justice. Still, these were the hardest pages to write in the thesis, far more difficult than the complex formulas, proofs, and technical details that fill the pages that follow. In fact, if these pages had been printed, they would have been covered with tears, as I slowly meditated on my whole life so far. At the same time, I wish to make clear that, although this thesis was written with two hands, it is, after all, the work of a lifetime: one shaped by the special people who have crossed my path since my earliest years. To all of you, from the depths of my heart: thank you, and may God bless you!

Sicut Philosophus dicit in principio Metaphysicæ, sapientis est ordinare. Cuius ratio est, quia sapientia est potissima perfectio rationis, cuius proprium est cognoscere ordinem.

"As the Philosopher says at the beginning of the Metaphysics, it belongs to the wise to order. The reason is that wisdom is the highest perfection of reason, whose proper task is to know order."

— S. Thomae de Aquino, Sententia libri Ethicorum, lib. I, lect. 1

List of Figures

List of Tables

List of Abbreviations and Acronyms

AI Artificial Intelligence

AR Autoregressive

BiLSTM Bidirectional Long Short-Term Memory

CNN Convolutional Neural Network

CPU Central Processing Unit

CS Client Selection

DP Differential Privacy

ECMTC Minimal **E**nergy **C**onsumption and **M**akespan FL Schedule under **T**ime **C**onstraint

ED Edge Device

FedAvg Federated Averaging

FL Federated Learning

FLOPS Floating-Point Operations Per Second

FLOPs Floating-Point Operations

FMA Fused Multiply-Add

FPOCC Floating-Point Operations per CPU Cycle per Core

GB Gigabyte

GDPR General Data Protection Regulation

GFLOP/s Giga Floating-Point Operations Per Second

GiB Gibibyte

GPU Graphics Processing Unit

gRPC Google Remote Procedure Call

IID Independent and Identically Distributed

IIoT Industrial Internet of Things

ILP Integer Linear Programming

IoT Internet of Things

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

LNS Large Neighborhood Search

LRU Least Recently Used

LSTM Long Short-Term Memory

LTE Long-Term Evolution

MB Megabyte

Mbps Megabits per Second

MEC Minimal **M**akespan and **E**nergy **C**onsumption FL Schedule

MetaCS-FL **Meta**heuristic-based **C**lient **S**election for **F**ederated **L**earning

MINLP Mixed-Integer Nonlinear Programming

ML Machine Learning

MTU Maximum Transmission Unit

NLP Natural Language Processing

Non-IID Non-Independent and Identically Distributed

NP Nondeterministic Polynomial Time

RAM Random Access Memory

RL Reinforcement Learning

RNN Recurrent Neural Network

RSS Resident Set Size

RTT Round-Trip Time

SGD Stochastic Gradient Descent

SIMD Single Instruction, Multiple Data

TPU Tensor Processing Unit

Wi-Fi Wireless Fidelity

Contents

1 Introduction

Driven by increasing data privacy concerns and regulations such as the GDPR[1], Federated Learning (FL) has emerged as a prominent distributed machine learning paradigm in which multiple participants, hereafter referred to as *Clients*, collaboratively train a model without sharing their raw data (1, 2).

In the standard FL setting, a central *Server* coordinates the training process. Each client performs computation locally using private data and transmits the resulting local model updates (*e.g.*, weights, gradients) to the server. The server aggregates the received updates to produce a new global model, which may be evaluated either centrally or at the client level. This iterative process continues until a predefined stopping criterion is met, such as reaching a target accuracy or a maximum number of communication rounds.

The client-server architecture of FL systems, commonly referred to as the *model-centric* approach (3), can be broadly categorized as *Cross-Device* or *Cross-Silo* FL. Cross-device FL involves a large number of mobile or edge devices, which are often unreliable, resource-constrained, and highly heterogeneous. Each device typically holds a small and potentially unbalanced local dataset, leading to non-independent and identically distributed (non-IID) data across clients (1). In contrast, cross-silo FL is conducted among a limited number of reliable organizations or data silos (*e.g.*, institutions or hospitals), which are characterized by more stable network connections, greater computational resources, and larger local datasets (4).

In cross-device FL, the choice of clients who participate in training strongly affects both system efficiency and learning behavior because clients differ in computational resources, communication conditions, energy availability, and local data distributions (5, 6, 7, 8, 9, 10, 11, 12). Selecting too many clients can increase energy consumption, communication overhead, and the likelihood of stragglers. Selecting too few clients, however, may reduce data representativeness, degrade model accuracy, and prolong convergence.

[1] https://gdpr-info.eu/

Moreover, repeatedly excluding the same clients may discourage long-term participation and weaken the collaborative nature of FL (13). Conversely, promoting diverse client participation across training rounds can improve robustness and help mitigate bias under heterogeneous data distributions (14, 15).

Despite the broad range of client selection strategies proposed in the literature (16), most existing approaches still treat participation as a binary decision: a client is either selected or excluded in a given communication round. Once selected, the client commonly processes a fixed or full local workload. This means that clients are often treated as indivisible units of computation, with limited control over how much data each selected client should process. Such a limitation is particularly relevant in heterogeneous cross-device FL, where devices differ in computation speed, energy availability, network quality, and reliability. Fine-grained workload control can help mitigate stragglers, reduce energy consumption on constrained devices, avoid overloading weaker clients, and allow more capable or data-rich clients to contribute more effectively.

A further limitation is that many existing approaches optimize only a narrow set of objectives, such as convergence speed, latency, or energy consumption. However, practical cross-device FL requires balancing several competing goals simultaneously, including training time, energy consumption, model performance, fairness of participation, data-distribution quality, and client reliability. In addition, the evaluation of existing approaches is often confined to image classification tasks, with limited validation across other data modalities, such as natural language processing. Finally, relatively few studies consider dynamic client availability, where clients may join after training has started, temporarily disconnect (*e.g.*, due to intermittent connectivity or battery depletion), and later return to the system (17, 18, 19, 20, 21, 22). These aspects are essential for assessing the applicability of client selection methods in realistic cross-device FL deployments.

The scope of this work is centralized, synchronous cross-device FL, targeting deployments in which a coordinating server manages heterogeneous, resource-constrained mobile, IoT, and edge devices through explicit communication rounds. The round boundary provides a common observation interval for comparing client performance and a defined point for scheduling participation and workload. Although asynchronous FL can reduce idle time and mitigate stragglers by using client updates as they arrive (23), it also introduces stale updates, which may require staleness-aware weighting, buffering, or rejection policies (23, 24). Moreover, asynchronous execution changes client selection from a round-level decision into a problem closer to admission control, concurrency regulation, and up-

date filtering (24). The proposed formulations do not target decentralized deployments or hard-real-time applications requiring decisions at millisecond or sub-millisecond scales. Therefore, asynchronous FL is a relevant complementary direction, but it is outside the scope of the present study.

1.1 Goals

The main goal of this thesis is to advance client selection in centralized, synchronous cross-device FL systems by designing optimization-based scheduling algorithms for time- and energy-aware workload allocation and extending them through a metaheuristic-based framework that also accounts for learning behavior, data heterogeneity, reliability, and fairness of participation.

The proposed approach adopts a task-oriented perspective in which training workloads are decomposed into smaller units, referred to as *tasks*. In this context, a task corresponds to a subset of local data, potentially including samples from one or more classes, such as object categories in images (25) or emotion labels in text (26). Tasks are independent from the scheduling perspective and may be defined with the same number of samples, but this common computational granularity does not imply identical statistical content: the samples and class proportions represented by different tasks may vary across clients. In this sense, client selection is formulated as a task scheduling problem, enabling the server to decide not only which clients participate but also how much workload each selected client processes and, in the metaheuristic-based framework, how that workload is distributed among locally available classes based on disclosed class-distribution statistics.

The first step in this direction is the design of two optimal scheduling algorithms, *MEC* and *ECMTC*, which control workload allocation while jointly optimizing time and energy consumption. These methods establish the benefits of moving beyond binary client selection and provide structured baselines for broader FL-specific scheduling formulations.

This work also introduces *MetaCS-FL*, a metaheuristic-based client selection framework that extends task-level scheduling beyond time and energy goals. MetaCS-FL balances makespan, energy consumption, participation diversity, class-distribution quality, reliability, and historical loss-based utility. It seeks to improve system efficiency and model performance while avoiding excessive work concentration on a small subset of clients.

Privacy and dynamic availability are also considered. Since data-distribution signals can guide class-aware client selection and workload allocation, the proposed framework

incorporates a privacy-preserving mechanism based on Differential Privacy (DP) (27), allowing the server to leverage noisy class-distribution statistics rather than exact local distributions. Furthermore, the evaluation includes dynamic scenarios involving late-joining and dropping/returning clients, assessing whether the proposed framework can maintain robust performance when the set of available clients changes during FL execution.

Accordingly, the thesis is structured around the following research questions (RQs):

RQ1 Client Participation and Workload Allocation: Compared to FedAvg (1) and state-of-the-art client selection strategies, how do MEC, ECMTC, and MetaCS-FL influence client participation and workload allocation across communication rounds? (*cf.* Section 6.2.1)

RQ2 System and Model Performance: How does MetaCS-FL influence system- and model-level performance, including test accuracy, convergence speed, total training time, and total energy consumption, under IID and non-IID data distributions across image and text classification domains? (*cf.* Section 6.2.2)

RQ3 Comparison between MEC, ECMTC, and MetaCS-FL: How does MetaCS-FL compare with the optimal schedulers MEC and ECMTC, which optimize workload allocation for time and energy but do not explicitly account for fairness, data-distribution quality, reliability, or loss-aware utility? (*cf.* Section 6.2.2)

RQ4 Privacy Trade-offs: How does DP-based client data-distribution disclosure affect representation, convergence, fairness, and system performance compared to the MetaCS-FL variant without DP-based protection? (*cf.* Section 6.2.3)

RQ5 Scalability and Server-Side Overhead: How does MetaCS-FL scale in terms of client selection time and server-side computational overhead as the number of tasks and available clients increases, compared to existing client selection methods? (*cf.* Sections 6.2.4 and 6.2.5)

RQ6 Parameter Sensitivity: How sensitive is MetaCS-FL to its main configuration parameters, including reselection triggers, objective-function weights, solution reuse, and metaheuristic search settings? (*cf.* Section 6.2.6)

RQ7 Late-Joining Clients: How do different fractions, arrival rounds, and performance characteristics of late-joining clients affect client selection decisions, workload allocation, convergence behavior, and system performance? (*cf.* Section 6.3.1)

RQ8 Intermittent-Availability Clients: How robust is MetaCS-FL when clients temporarily become unavailable, later return, or fail to complete assigned workloads under intermittent-availability and completion-failure conditions? (*cf.* Section 6.3.2)

1.2 Contributions

Based on the goals outlined above, the main contributions of this thesis are:

I. The formulation of client selection in synchronous cross-device FL as a task scheduling problem with fine-grained control over the amount of local training data assigned to each selected client, while distinguishing common scheduling granularity from the potentially non-IID statistical content of the underlying data;

II. The design of *MEC* and *ECMTC* (28), two optimal workload scheduling algorithms that jointly optimize training time and energy consumption while controlling task allocation across heterogeneous clients;

III. The development of *MetaCS-FL* (29), the main contribution of this thesis, as a flexible and event-driven metaheuristic-based framework for client selection and workload scheduling in synchronous cross-device FL;

IV. A unified multi-objective scheduling formulation for MetaCS-FL that jointly considers makespan, energy consumption, client participation diversity, class-distribution quality, reliability, and historical loss-based utility;

V. The integration of a DP-based privacy-preserving mechanism (27), allowing the server to employ noisy client class-distribution statistics for class-aware selection and workload allocation without accessing exact local distributions;

VI. The introduction of reliability-aware workload control, which adjusts client task capacities according to availability and completion behavior, reducing the risk of assigning excessive workloads to unstable clients.

1.3 Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces the key concepts underlying this work. Chapter 3 reviews state-of-the-art client selection strategies in FL and positions this work with respect to existing approaches. Chapter 4 presents *MEC* and *ECMTC*, two optimal workload scheduling algorithms for time- and energy-aware client selection. Chapter 5 introduces *MetaCS-FL*, including its system model, optimization problem, reliability-aware workload control, privacy-preserving data-distribution disclosure, and metaheuristic-based selection procedure. Chapter 6 describes the experimental setup and discusses the evaluation results, including an analysis of performance, privacy, scalability, sensitivity, server-side overhead, and dynamic client availability. Finally, Chapter 7 concludes the thesis and outlines directions for future research.

2 Background

This chapter provides the background for the client selection problem addressed in this work. It first reviews Machine Learning (ML) fundamentals, including supervised learning, neural networks, training, testing, data distribution, and generalization, which explain how models are trained and evaluated (Section 2.1). It then introduces Federated Learning (FL), covering the collaborative training process, FedAvg, cross-silo and cross-device settings, statistical and system heterogeneity, synchronous and asynchronous execution, privacy and security, client selection, FL experimentation frameworks, and evaluation metrics (Section 2.2). Finally, the chapter presents task scheduling, dynamic programming, and metaheuristics, which provide the optimization background for modeling FL client selection as a scheduling problem (Sections 2.3, 2.4, and 2.5).

2.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence focused on developing models that learn patterns from data and use them to make predictions on previously unseen examples (30, 31). Rather than relying on explicitly programmed decision rules, an ML algorithm estimates a predictive function from observed data. This function is represented by a model whose parameters are adjusted during training according to a specified optimization objective.

2.1.1 Supervised Learning

In supervised learning, the training data consist of input-output pairs. Each input sample is associated with a target value, which may be a discrete class label in classification problems or a continuous value in regression problems (30, 31). Given a training dataset $\mathcal{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{|\mathcal{D}|}$, where $\mathbf{x}_j$ denotes an input sample and y_j denotes its corresponding target, the goal is to learn a model $f_{\mathbf{w}}$, parameterized by $\mathbf{w}$, that maps each input $\mathbf{x}_j$ to

a prediction $f_{\mathbf{w}}(\mathbf{x}_j)$.

The discrepancy between the model prediction and the true target is measured by a loss function $\ell(\cdot)$. Training consists of finding parameters $\mathbf{w}$ that minimize the empirical loss over the available training data (30, 32):

$$\min_{\mathbf{w}} \; L(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_j, y_j) \in \mathcal{D}} \ell(f_{\mathbf{w}}(\mathbf{x}_j), y_j), \tag{2.1}$$

where $L(\mathbf{w})$ denotes the average training loss. In classification problems, a common choice is the cross-entropy loss, which penalizes incorrect probability assignments to the true class (32). The optimization problem in Eq. 2.1 is solved using iterative gradient-based methods, such as stochastic gradient descent (SGD) and its variants (33, 34, 35). These methods update the model parameters in directions that reduce the loss, typically using gradients computed from individual samples or mini-batches of the training data.

A key goal of ML is generalization, *i.e.*, the ability of a model to perform well on data that were not used during training (31). For this reason, models are typically evaluated on a separate test set after training. In classification problems, a common evaluation metric is accuracy, defined as the fraction of correctly classified examples. Training loss and test performance are related but not equivalent: a model may achieve low loss on the training data while performing poorly on unseen examples if it overfits the training set (30, 31).

The quality of the learned model depends on several factors, including model architecture, the optimization procedure, the amount of available data, and the data distribution. When the training data are representative of the target distribution, the model is more likely to generalize well. Conversely, if the training data are biased, imbalanced, or incomplete, the resulting model may perform poorly on underrepresented patterns or classes. This relationship among model design, data characteristics, and generalization becomes especially relevant in distributed and federated settings, where the global dataset is spread across multiple participants.

2.1.2 Neural Networks

Neural networks are a widely used category of ML models composed of layers of parameterized transformations. Each layer receives an input representation, applies a transformation defined by trainable parameters, and produces an output representation that is

passed to the next layer (32). A basic layer can be written as

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{2.2}$$

where $\mathbf{h}^{(l)}$ denotes the representation produced by layer l, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are trainable weights and biases, and $\sigma(\cdot)$ is a nonlinear activation function. The first layer receives the input data, whereas the final layer produces the model prediction. The parameters of all layers are learned by minimizing a loss function, such as the empirical loss defined in Eq. 2.1.

Deep neural networks stack multiple layers, allowing the model to learn hierarchical representations of the input data (32). Lower layers typically capture simple patterns, while deeper layers combine these patterns into more abstract and task-specific representations. This form of representation learning is useful because it reduces the need for hand-crafted features and enables the model to adapt its internal representations to the target machine learning task.

2.1.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural network architectures designed to process grid-structured data, particularly images (36, 32). Unlike fully connected layers, in which each output unit depends on all input units, convolutional layers apply learnable filters to local regions of the input. This design relies on two key principles: local connectivity and parameter sharing. Local connectivity allows the model to focus on spatially neighboring pixels, while parameter sharing enables the same filter to detect a pattern across different regions of the image.

Given an input image or feature map, a convolutional layer slides a set of kernels over the spatial dimensions of the input. Each kernel produces a feature map that indicates where a learned pattern is present. As convolutional layers are stacked, the network can learn progressively more complex visual features. Early layers may capture edges, corners, and textures, whereas deeper layers may represent object parts or class-specific structures (32).

CNNs are often combined with pooling, normalization, and fully connected layers. Pooling reduces the spatial resolution of feature maps and makes the learned representation more robust to small translations. Normalization layers can stabilize and accelerate training. Fully connected layers near the output transform the learned representation into class scores. In classification problems, these scores are typically converted into

class probabilities using a softmax function, and the model is commonly optimized using cross-entropy loss (32).

CNNs are relevant to this work because they are commonly used for image-based tasks in Federated Learning (FL). Their computational cost depends on factors such as the number of layers, filters, and parameters, as well as the input size, batch size, and number of local epochs. Consequently, the model architecture directly affects the amount of computation performed by each client. In heterogeneous FL settings, clients may train the same CNN at different speeds due to differences in hardware capabilities, memory bandwidth, and communication conditions.

2.1.2.2 RNNs, LSTMs, and attention-based models

While CNNs are well suited for spatial data, sequential data require models capable of representing dependencies across ordered elements. Examples include text, speech, time series, and event sequences. Recurrent Neural Networks (RNNs) address this setting by processing one sequence element at a time while maintaining a hidden state that summarizes information from previous elements (32). This recurrent structure allows the model to incorporate contextual information when producing predictions.

However, standard RNNs may struggle to learn long-range dependencies due to vanishing and exploding gradients. Long Short-Term Memory (LSTM) networks were introduced to mitigate this limitation by using gating mechanisms that regulate the flow of information through the sequence (37). These gates control which information should be stored, forgotten, and exposed at each step, allowing the model to retain useful information over longer intervals.

Bidirectional LSTMs extend this idea by processing the sequence in two directions: one from the beginning to the end and another from the end to the beginning. This allows each position in the sequence to be represented using both past and future context. Such bidirectional processing is particularly useful in text classification, where the meaning of a token may depend on words that appear before and after it.

Attention mechanisms allow sequence models to assign different levels of importance to different parts of the input (38). Instead of compressing the entire sequence into a single fixed representation, attention computes weights over the sequence elements and combines them according to their relevance to the prediction. This mechanism is useful when only certain words or sequence elements are strongly related to the target label.

Transformers extend attention-based modeling by replacing recurrent sequence processing with self-attention layers that directly represent relationships among all positions in a sequence (39). Given that Transformers do not process sequence elements strictly one at a time, their computations can be parallelized more effectively during training, while positional encodings preserve information about element order. This architecture has become particularly prominent in natural language processing because it can represent long-range dependencies without relying on recurrent hidden states. Nevertheless, standard self-attention introduces computation and memory costs that grow quadratically with sequence length, which may limit the local training of large Transformer models on resource-constrained devices.

RNNs, LSTMs, attention mechanisms, and Transformers are relevant in FL because many real-world applications involve sequential data held by different clients, such as user-typed text, sensor readings, mobility traces, and other time-dependent records. As with CNNs, the architecture influences the local computational cost of training and testing. At the same time, the distribution of local sequences across clients affects the learning signal contributed by each participant.

2.1.3 Training, Testing, Data Distribution, and Generalization

Training, testing, data distribution, and generalization are closely related concepts in ML. Training is the process of adjusting model parameters to reduce the loss on available data, whereas testing estimates how well the trained model performs on data that were not used during optimization (31). In centralized ML, both processes are typically defined with respect to data stored and managed in a single location. This setting makes it possible to inspect the global data distribution, construct training and testing splits, and apply preprocessing uniformly.

In ML, minimizing the training loss is only part of the objective; the model must also learn patterns that generalize to unseen examples. When training and test data are sampled from similar distributions, the model can use the patterns learned during training to make accurate predictions on test examples. However, when the training data are biased, incomplete, imbalanced, or not representative of the target distribution, the model may generalize poorly (30, 31). In such cases, low training loss does not necessarily imply high performance on unseen test data.

In classification problems, the class distribution is particularly relevant. A balanced dataset contains similar numbers of samples from each class, whereas an imbalanced

dataset contains substantially more samples from some classes than from others. If a model is trained primarily on frequent classes, it may learn decision boundaries biased toward those classes and perform poorly on rare or underrepresented classes. In centralized ML, class imbalance can be observed directly from the global dataset and addressed using preprocessing, sampling, weighting, or other mitigation strategies. In distributed settings, however, class imbalance is harder to identify and mitigate because no single participant may have access to the full data distribution.

This challenge is one example of a broader data heterogeneity problem, which is commonly described through the distinction between independent and identically distributed (IID) and non-IID data. In the IID setting, each local dataset can be viewed as a representative sample of the global distribution. In the non-IID setting, different participants may hold data with different class distributions, different numbers of samples, or different statistical properties (40, 14).

2.2 Federated Learning

Federated Learning (FL) is a distributed machine learning paradigm in which multiple clients collaboratively train a shared model while keeping their raw data local (1, 40). Instead of collecting all training samples in a central repository, each client performs computation on its own local dataset and communicates model updates to a coordinating server. The server aggregates these updates to produce a new global model, which is then redistributed to the clients for the next training round, also known as a communication round. This process allows learning from decentralized data while reducing the need to transfer potentially sensitive or large-scale datasets (1, 40).

The formulation introduced by McMahan et al. (1) considers a finite-sum optimization problem of the form

$$\min_{\mathbf{w}} \; F(\mathbf{w}), \qquad F(\mathbf{w}) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(\mathbf{w}), \tag{2.3}$$

where K is the number of clients, $n_k = |\mathcal{D}_k|$ is the number of training samples held by client k, and $n = \sum_{k=1}^{K} n_k$ is the total number of samples across all clients. The local objective of client k is defined as

$$F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_k} \ell(f_{\mathbf{w}}(\mathbf{x}_i), y_i), \tag{2.4}$$

where $\mathcal{D}_k$ denotes the local dataset of client k, $\ell(\cdot)$ is the loss function, and $f_{\mathbf{w}}$ is the

model parameterized by $\mathbf{w}$. Therefore, the global FL objective is a weighted average of the local objectives, with each client weighted according to the size of its local dataset (1, 40).

A widely used baseline algorithm for solving this objective is Federated Averaging (FedAvg) (1). FedAvg extends stochastic gradient descent by allowing selected clients to perform multiple local optimization steps before communicating with the server. At each communication round r, the server sends the current global model $\mathbf{w}_r$ to a subset of clients randomly selected from the pool of available clients. Each selected client initializes its local model with $\mathbf{w}_r$ and updates it using its local data for one or more local epochs. After local training, the clients send their updated model parameters back to the server. The server then computes a weighted average of the received client models:

$$\mathbf{w}_{r+1} = \sum_{k \in \mathcal{S}_r} \frac{n_k}{\sum_{j \in \mathcal{S}_r} n_j} \mathbf{w}_{r+1}^k, \tag{2.5}$$

where $\mathcal{S}_r$ is the set of clients selected at round r and $\mathbf{w}_{r+1}^k$ denotes the model obtained by client k after local training. The weighting factor accounts for the number of samples held by each participating client, so that clients with larger local datasets contribute proportionally more to the aggregated model (1).

The FedAvg procedure can be summarized as follows (Algorithm 1):

Algorithm 1 Federated Averaging (FedAvg)

Initialize global model parameters $\mathbf{w}_0$
for each communication round $r = 1,2,\ldots$ **do**
    Server selects a subset of clients $\mathcal{S}_r$
    Server sends $\mathbf{w}_r$ to each client $k \in \mathcal{S}_r$
    for each selected client $k \in \mathcal{S}_r$ in parallel **do**
        Initialize local model $\mathbf{w}_r^k \leftarrow \mathbf{w}_r$
        Update $\mathbf{w}_r^k$ using local data $\mathcal{D}_k$ for E local epochs
        Send updated local model parameters $\mathbf{w}_{r+1}^k$ to the server
    Server aggregates the received local model parameters using Eq. 2.5

The main advantage of FedAvg is that it reduces communication frequency by performing several local updates between server aggregations. By decreasing the number of communication rounds, FedAvg addresses one of the main bottlenecks in FL, particularly when clients are connected through unreliable or low-bandwidth networks (1, 40).

Although FL is related to traditional distributed learning, the two settings rely on different assumptions. In conventional distributed learning, data are often distributed across workers mainly to parallelize computation, and the data can usually be assumed

to be balanced and approximately independent and identically distributed (IID). In FL, however, data are naturally generated and stored by different clients. As a result, local datasets may be unbalanced, statistically heterogeneous, and unavailable at different times (40, 14). These characteristics make optimization and evaluation more challenging than in centralized or datacenter-based distributed learning.

2.2.1 Cross-Silo and Cross-Device Systems

FL systems are commonly categorized according to the type, scale, and reliability of the participating clients. Two main categories are cross-silo FL and cross-device FL (40). Although both settings follow the same general principle of training a shared model from decentralized data, they differ substantially in terms of client population, availability, computational capacity, communication constraints, and deployment assumptions.

In cross-silo FL, the clients are usually organizations or institutional data holders, such as hospitals, banks, companies, universities, or research centers. Compared with cross-device FL, the number of clients is usually smaller, but each client may hold a larger, more stable, and more curated dataset (40). Cross-silo clients are also generally more reliable and have stronger computational and communication resources. Since the participating entities are often known in advance, cross-silo FL may involve more stable participation across training rounds and stronger coordination among the parties.

Cross-silo FL is particularly relevant when different institutions want to collaboratively train a model without directly sharing their raw data. This may occur because of privacy requirements, legal restrictions, regulatory constraints, commercial interests, or data governance policies. For example, hospitals may collaborate to train a medical diagnosis model while keeping patient records local, or financial institutions may jointly train fraud detection models without exposing private customer data. In this setting, communication constraints may still be relevant, but the main challenges often involve data heterogeneity across institutions, trust among participants, access control, auditing, and fairness in the contribution and benefit of each organization (40, 14).

In cross-device FL, the clients are typically edge devices, such as smartphones, tablets, wearable devices, sensors, or Internet-of-Things (IoT) devices. This setting may involve a very large population of clients, often ranging from thousands to millions of devices. However, only a small fraction of these clients is usually available and selected to participate in each communication round (40). Client availability depends on practical conditions such as network connectivity, battery level, device usage, and local resource constraints. As

a result, cross-device FL must be designed to tolerate partial participation, intermittent connectivity, and frequent client dropout.

Resource heterogeneity is another defining characteristic of cross-device FL. Devices may have different processors, memory capacities, energy constraints, and communication bandwidths. Consequently, the time required to perform local training and transmit model updates can vary significantly across clients. These differences may lead to stragglers, delayed updates, or incomplete participation in a given round. For this reason, client selection, communication efficiency, compression techniques, and robustness to unreliable participation are central concerns in cross-device FL (40, 14). This setting is common in applications where data are generated directly by users or devices, such as keyboard prediction, mobile personalization, activity recognition, and sensor-based learning.

The distinction between cross-silo and cross-device FL directly affects the design of the learning system. Cross-silo FL usually focuses on collaboration among a smaller number of stable participants, where institutional data heterogeneity, governance, reliability, and fairness are central concerns. Cross-device FL, in contrast, must handle large-scale client populations, limited local resources, frequent client dropout, and highly variable availability. In both cases, the global model is learned from decentralized data, but the practical constraints and system assumptions differ substantially.

2.2.2 Statistical and System Heterogeneity

Client heterogeneity is a major challenge in FL, particularly in cross-device settings, where clients are numerous, resource-constrained, and frequently unavailable. Unlike centralized ML, where data and computation are typically managed within a single environment, FL involves multiple clients with potentially different data distributions, computational capabilities, and communication resources (40, 14, 41).

Statistical heterogeneity refers to differences in the data distributions observed by different clients. Since each client collects data according to its own context, behavior, environment, or population, local datasets are often non-IID. For example, different clients may hold different class distributions, different numbers of samples, or data with different statistical properties. Such heterogeneity can cause local updates to point in different optimization directions, slowing convergence or reducing the quality of the global model. This distinction is independent of granularity adopted by a scheduling model: two task units may contain the same number of samples while representing different classes or distributions. Thus, equal-sized units must not be interpreted as IID data.

System heterogeneity refers to differences in the computational and communication capabilities of clients. Clients may differ in processing power, memory capacity, network bandwidth, energy availability, and local training speed. These differences can lead to stragglers, delayed updates, or incomplete participation in a training round. In synchronous FL, slow clients may increase the duration of each round because the server waits for the updates from all selected clients before performing model aggregation.

Statistical and systems heterogeneity are often related. For example, excluding slow or unreliable clients may reduce computation and communication costs, but it can also bias the learning process if those clients hold data from underrepresented classes or populations. Therefore, the design of FL systems must balance model performance, computational efficiency, communication costs, and data coverage.

These challenges depend on the execution model adopted by the FL system. In synchronous protocols, heterogeneity mainly appears through stragglers, round delays, and incomplete participation. In asynchronous protocols, the same heterogeneity may appear through stale updates, uneven contribution frequency, and the need to regulate which updates are incorporated into the global model.

2.2.3 Synchronous and Asynchronous Execution

FL systems can differ not only in client scale and deployment setting, but also in how client updates are coordinated and aggregated. Two common execution models are synchronous and asynchronous FL. This distinction is especially relevant in cross-device FL, where clients may have heterogeneous resources, intermittent availability, and variable communication conditions (40). However, cross-device FL does not impose an execution model; it can be implemented using synchronous, asynchronous, or hybrid protocols.

In synchronous FL, training progresses in communication rounds. At each round, the server selects a subset of clients, sends the current global model, waits for local training, and aggregates the updates received for that round (1, 40). This preserves a clear round structure: selected clients train from the same global model version, and aggregation is performed over updates associated with the same logical round. Therefore, synchronous FL naturally supports round-level decisions such as client selection, workload assignment, deadline enforcement, fairness control, and comparison among selected clients. In the centralized setting, the common round boundary simplifies coordination, as client selection, local training, and aggregation occur at defined points. It also provides a consistent observation interval for refreshing client profiles and comparing computation, energy, and

communication measurements before the next selection decision. Nevertheless, concentrating these responsibilities at the server may create a coordination and computation bottleneck as the client population grow.

The drawback of synchronous FL is its sensitivity to stragglers, a term used here in its FL-specific sense to denote selected clients whose local computation or communication delays or threatens the completion of a synchronous round. Since aggregation occurs at round boundaries, slow clients can increase round duration in heterogeneous cross-device environments (40, 42). This FL-specific use differs from the established meaning in optimistic distributed simulation, where a straggler is a late event or message whose timestamp precedes the receiver's current simulation time and may trigger a rollback. However, synchronous FL does not require the server to wait indefinitely. In deadline-based synchronous protocols, the server can define a maximum waiting time and aggregate only the updates received before the deadline, limiting the impact of slow or unavailable clients while preserving round-based coordination. In the approach considered in this thesis, an update returned after the round deadline is discarded and excluded from that round's aggregation. The missed completion is recorded in the client's history and incurs a completion penalty in the reliability mechanism proposed. Consequently, the client's reliability score may decrease and restrict its admissible task capacities in later selections, reducing its future workload without permanently excluding it from participation.

In asynchronous FL, clients do not need to complete training within a shared synchronization round. Instead, the server can use updates as they arrive (23). The number of clients training at a given time is usually constrained by client availability and controlled through admission or concurrency limits (24). This can reduce idle time, mitigate stragglers, and handle irregular client availability. However, asynchronous FL introduces stale updates, as clients may train using outdated models and return updates after newer updates have been incorporated (23, 24). Handling staleness may require mechanisms such as staleness-aware weighting, bounded-delay aggregation, buffered aggregation, or update rejection policies.

Overall, synchronous FL favors round-level coordination and client-management decisions, whereas asynchronous FL favors continuous update processing at the cost of staleness management. This distinction directly affects how client selection is formulated. In synchronous FL, client selection is naturally formulated as a round-level decision, where the server determines the participating clients before local training starts and can reason about the expected duration, workload, and composition of the selected group. In asyn-

chronous FL, client selection can still be useful, but it usually shifts toward admission control, concurrency regulation, update acceptance, or staleness-aware filtering.

2.2.4 Privacy and Security

Although FL keeps raw data on clients, this design does not, by itself, guarantee privacy. Even when clients share only model updates, gradients, or auxiliary statistics, an adversary may still infer sensitive properties of local datasets through inference attacks (40). This risk becomes particularly relevant when the exchanged information reflects local data distributions, rare classes, or client-specific patterns. Therefore, FL systems often require additional privacy-preserving mechanisms to limit what can be learned from the information exchanged during training and coordination.

Two common mechanisms are Differential Privacy (DP) and Secure Aggregation. DP provides privacy guarantees by adding calibrated noise to the information released by a client or by the training process (43, 44). In FL, DP can be applied on the client side. For example, by perturbing model updates or auxiliary statistics before they are sent to the server or centrally during aggregation. The privacy parameter ϵ controls the strength of the guarantee, with smaller values providing stronger privacy but typically reducing utility.

Secure Aggregation, in contrast, is a cryptographic protocol that allows the server to compute an aggregate over client updates without observing each individual contribution (45). This mechanism is useful when the server needs the combined update from several clients while hiding the individual updates. However, Secure Aggregation usually requires interactive coordination among the participating clients and is closely tied to the training rounds in which aggregation occurs.

In this work, privacy is relevant because the server may require information about local data distributions during client selection. Directly exposing these distributions could reveal sensitive information about a client's data, especially when some classes are rare or are strongly associated with specific clients. Therefore, auxiliary statistics used for client selection should be disclosed in a privacy-preserving manner, for example, by perturbing local class-count histograms before they are shared with the server.

2.2.5 Client Selection

Client selection is the process of deciding which clients are allowed or prioritized to contribute to the FL process. In round-based synchronous FL, this usually means choosing the subset of clients that participate in each communication round. Since involving all clients in every round is often impractical, especially in cross-device settings with large client populations, FL algorithms commonly train with only a subset of available clients at each round (1, 40). In asynchronous FL, this role is less tied to fixed communication rounds and is more closely related to deciding which client updates are accepted, how many clients train concurrently, and how stale updates are handled (23, 24).

Random client selection is simple and widely used as a baseline because it does not require detailed information about client resources or data distributions (1). However, in heterogeneous FL settings, random selection may lead to inefficient or biased training. For example, selected clients may have limited computational or communication resources, increasing the duration of a round. At the same time, random selection may repeatedly exclude clients that hold rare or underrepresented data, which can affect model performance and fairness.

More informed client selection strategies attempt to improve FL efficiency by considering system-related or data-related criteria. System-aware selection prioritizes clients according to resource conditions such as computation speed, network bandwidth, availability, or expected completion time (7). Data-aware or utility-aware selection, in contrast, prioritizes clients whose updates are expected to be more useful for improving the global model (46). These strategies aim to improve convergence, reduce training time, and increase the usefulness of the selected client updates.

Client selection must therefore balance competing objectives. Selecting only low-latency clients can reduce computation and communication delays, but it may bias training if those clients are not representative of the global data distribution. Conversely, selecting clients solely based on data utility may improve model performance but increase round duration when high-utility clients have limited resources. Thus, client selection is closely related to both statistical and system heterogeneity (40, 14, 41).

2.2.6 Experimentation Frameworks

FL frameworks provide abstractions to implement, simulate, and deploy FL workloads without building the infrastructure for client orchestration, model exchange, aggregation,

and communication. These frameworks are useful because FL experiments must capture both learning behavior, such as convergence and model accuracy, and system-level aspects, such as client availability, communication overhead, heterogeneity, and scalability.

Several frameworks support FL experimentation. TensorFlow Federated (TFF) focuses on decentralized computations using TensorFlow-based models (47), FedML provides a research library and benchmark for FL simulation and deployment across mobile, edge, and cloud environments (48), and NVIDIA FLARE targets production-oriented cross-silo FL deployments (49). FedScale, in turn, focuses on benchmarking FL at scale by combining model performance with system-level aspects such as client heterogeneity, availability, and execution behavior (50). These frameworks differ in their goals, ranging from algorithmic experimentation and benchmarking to scalable and secure deployment.

Other frameworks focus on improving FL experimentation and sustainability-oriented analysis. Fluke (51) is a lightweight Python framework for rapid prototyping and evaluation of FL algorithms. It provides a simulated FL environment, several algorithms and datasets, and abstractions for configuring clients, servers, communication, and evaluation. Its main goal is to reduce implementation overhead when testing new FL algorithms, especially when the focus is on learning behavior rather than deployment infrastructure.

FedSynthesis (52) focuses on sustainability-aware FL experimentation by integrating carbon-aware measurements into the FL execution process. It uses measured client-side emissions to support carbon-aware client selection, allowing the server to prioritize clients expected to train with lower carbon footprint in future rounds. Thus, FedSynthesis is relevant as an example of how FL frameworks can incorporate sustainability metrics into experimental workflows and selection decisions.

Flower is a general-purpose and framework-agnostic FL framework designed to support both research and practical experimentation (53). It allows existing centralized ML pipelines to be adapted to FL with limited changes and supports multiple backends, including PyTorch, TensorFlow, JAX, Hugging Face, NumPy, scikit-learn, and XGBoost. In Flower, clients define local training and evaluation behavior, while the server coordinates FL rounds by selecting clients, distributing model parameters, collecting updates, and applying an aggregation strategy.

A key feature of Flower is the separation between clients and strategies. Clients encapsulate local computation over private data, whereas strategies define the server-side logic for client sampling, round configuration, aggregation, and evaluation. This design makes Flower suitable for implementing custom client selection mechanisms and studying

heterogeneous FL scenarios in which clients may differ in data distribution, computational capacity, availability, energy consumption, and execution time.

2.2.7 Evaluation Metrics

Evaluating FL systems requires metrics that capture model quality, convergence speed, training time, energy consumption, client selection overhead, and fairness of participation.

Rounds to Accuracy. Let R_x denote the first communication round in which the global model reaches a target test accuracy x. The number of rounds required to reach this target accuracy is denoted by

$$\mathrm{RoA@}x = R_x \tag{2.6}$$

Makespan. Let $\mathcal{S}_r$ denote the set of clients selected in round r, and let δ_i denote the time spent by selected client i in that round. Since a synchronous FL round is limited by the slowest selected client, the makespan of round r is defined as

$$\mathrm{M}_r = \max_{i \in \mathcal{S}_r} \delta_i \tag{2.7}$$

The total training time until the target accuracy is reached is then defined as

$$\mathrm{M}_{\text{total}} = \sum_{r=1}^{R_x} \mathrm{M}_r \tag{2.8}$$

Energy Consumption. Let ϵ_i denote the energy consumed by selected client i in round r. The energy consumed in round r is computed as the sum of the energy spent by the selected clients:

$$\Sigma_r = \sum_{i \in \mathcal{S}_r} \epsilon_i \tag{2.9}$$

The total training energy is then defined as

$$\Sigma_{\text{total}} = \sum_{r=1}^{R_x} \Sigma_r \tag{2.10}$$

Client Selection Time. When client selection is performed at each round, the selection procedure itself may introduce computational overhead. The total client selection time is therefore defined as

$$\mathrm{T}_{\text{cs}} = \sum_{r=1}^{R_x} \mathrm{T}_{\text{cs},r}, \tag{2.11}$$

where $T_{cs,r}$ is the time spent selecting clients in round r.

Selection Fairness. Fairness of client selection can be measured using Jain's Fairness Index (54). Let s_i denote the number of times client i is selected during training, and let n be the total number of clients. The selection fairness score is given by

$$S_{\text{fair}} = \frac{\left(\sum_{i=1}^{n} s_i\right)^2}{n \cdot \sum_{i=1}^{n} s_i^2}, \tag{2.12}$$

with values closer to 1 indicating more uniform client participation. Here, fairness refers specifically to uniformity in the number of selections across clients and does not measure equality of workload, energy use, data representation, or model quality.

Test Accuracy. Finally, if client i has n_i^{test} test samples and achieves local test accuracy a_i, the global test accuracy is computed as the weighted mean

$$\text{Acc} = \frac{\sum_{i=1}^{n} n_i^{\text{test}} \cdot a_i}{\sum_{i=1}^{n} n_i^{\text{test}}} \tag{2.13}$$

2.3 Task Scheduling Problems

Scheduling concerns the allocation of scarce or limited resources to a set of activities while satisfying system constraints and pursuing one or more objectives (55, 56, 57). A scheduling problem is therefore characterized by three main elements: the activities to be performed, the resources available to perform them, and the objective according to which candidate schedules are evaluated (57). In computing systems, activities may represent jobs, operations, requests, data chunks, or computational tasks, whereas resources may represent processors, machines, servers, accelerators, or distributed devices.

The characteristics of both activities and resources determine the structure of the scheduling problem. Tasks may differ in processing time, priority, release time, deadline, or resource requirements, and they may be independent or subject to precedence constraints. Similarly, resources may differ in processing capacity, availability, memory, energy consumption, communication performance, or reliability. A scheduling decision must account for these characteristics while determining which resource executes each task and, when relevant, the order and time at which tasks are executed.

2.3.1 Scheduling Fundamentals

Scheduling problems may be formulated either as decision problems or as optimization problems (57). A decision problem asks whether there exists a schedule satisfying all imposed constraints, such as whether every task can be completed before its deadline. An optimization problem additionally seeks a feasible schedule that minimizes or maximizes a given performance metric. Typical objectives include minimizing the makespan, total completion time, energy consumption, monetary cost, or communication overhead, and maximizing throughput, resource utilization, or fairness (55, 58).

Let $\mathcal{J} = \{1,\ldots,J\}$ denote a set of tasks and let $\mathcal{M} = \{1,\ldots,P\}$ denote a set of machines or processing agents. Depending on how processing times vary across resources, machines are commonly classified as identical, uniform, or unrelated. In identical machine scheduling, every task has the same processing time on every machine. In uniform-machine scheduling, machines operate at different, but task-independent, speeds. In unrelated-machine scheduling, the processing time of a task depends on the specific machine to which it is assigned (55).

Consider independent, in the scheduling sense of having no precedence constraints, and non-preemptive tasks, with no communication costs or release-time restrictions. Let p_{jk} denote the processing time of task j when executed by machine k, and let $a_{jk} \in \{0,1\}$ indicate whether task j is assigned to machine k. Each task must be assigned to exactly one machine:

$$\sum_{k\in\mathcal{M}} a_{jk} = 1, \qquad \forall j \in \mathcal{J} \tag{2.14}$$

Under these assumptions, the total load assigned to machine k is

$$L_k = \sum_{j\in\mathcal{J}} a_{jk} \cdot p_{jk} \tag{2.15}$$

The makespan is the required time of the last machine to finish its assigned workload:

$$C_{\max} = \max_{k\in\mathcal{M}} L_k \tag{2.16}$$

The corresponding optimization problem is

$$\min_{\{a_{jk}\}} C_{\max} \tag{2.17}$$

Makespan minimization attempts to prevent a heavily loaded or slow resource from

becoming the bottleneck of the schedule. In heterogeneous systems, however, balancing the number of assigned tasks is not necessarily sufficient, because the same workload may incur different processing times on different resources.

When several objectives are relevant, they may be combined in different ways. A weighted or scalarized objective combines multiple metrics into a single value. Lexicographic optimization orders the objectives by importance and optimizes them sequentially. Alternatively, secondary objectives may be transformed into constraints, such as minimizing energy consumption subject to a maximum execution-time bound. The selected formulation determines how trade-offs between candidate schedules are evaluated (57).

Many scheduling problems are combinatorial and NP-hard (59, 56). Consequently, the number of possible schedules may grow rapidly with the numbers of tasks and resources. The difficulty generally increases when resources are heterogeneous, tasks have resource-dependent costs, or multiple constraints and objectives must be considered simultaneously. Depending on the problem size and structure, solutions may therefore rely on exact algorithms, approximation algorithms, heuristics, or metaheuristics.

2.3.2 Client Selection as a Task Scheduling Formulation

In this work, client selection and workload allocation for synchronous FL are modeled jointly as a scheduling problem. The clients act as heterogeneous processing resources, while the workload consists of units of local training effort. These workload units are represented as data slices, with each task corresponding to a chunk of local data used to form mini-batches to be processed by the selected clients. The tasks are assumed to be equivalent in scheduling granularity, independent, and atomic: each task represents the same configured amount of local data, although its samples and class composition may differ across clients. Here, independence denotes the absence of precedence constraints, and atomicity means that a data slice cannot be divided below the configured scheduling unit. For MEC and ECMTC, execution-time and energy costs depend on the receiving client and assigned workload rather than on the semantic contents of individual slices (8, 60, 57). MetaCS-FL additionally maintains per-class task counts and uses them to evaluate the class coverage and balance produced by each workload allocation (29).

Let $\mathcal{C}_r$ denote the set of clients eligible to participate in round r, and let t denote the total user-defined workload to be distributed. For each client $i \in \mathcal{C}_r$, let $\widetilde{\mathtt{AC}}_{i,r}$ denote the finite set of feasible assignment capacities in that round. These capacities may differ across clients due to local data availability, processing capability, memory limitations,

reliability requirements, or other resource constraints. The value zero is included as the option of not selecting the client: $0 \in \widetilde{\mathtt{AC}}_{i,r}$.

Let x_i denote the number of tasks assigned to client i. In MetaCS-FL, this total is further distributed among client i's locally available classes according to its available class-count vector, which may contain exact or DP-perturbed values. Consequently, the scheduling decision can account for both workload amount and class composition without transferring raw samples to the server. The assignment decisions must satisfy

$$x_i \in \widetilde{\mathtt{AC}}_{i,r}, \qquad \forall i \in \mathcal{C}_r, \tag{2.18}$$

and the complete workload must be distributed:

$$\sum_{i \in \mathcal{C}_r} x_i = t \tag{2.19}$$

Client selection and workload allocation are therefore represented by the same decision variable:

$$x_i = \begin{cases} 0, & \text{if client } i \text{ is not selected,} \\ > 0, & \text{if client } i \text{ is selected.} \end{cases} \tag{2.20}$$

The set of selected clients in round r is consequently

$$\mathcal{S}_r = \{i \in \mathcal{C}_r \mid x_i > 0\} \tag{2.21}$$

For each client, a cost function can associate every feasible workload with a predicted execution time, energy consumption, or another scheduling metric. For example, if $c_{i,r}^{\text{time}}(x_i)$ denotes the time required by client i to process x_i tasks, then the duration of a synchronous round is determined by its slowest selected client:

$$C_{\max,r} = \max_{i \in \mathcal{C}_r} c_{i,r}^{\text{time}}(x_i) \tag{2.22}$$

This client is commonly referred to as a straggler (8, 57). Scheduling the workload therefore provides a mechanism for reducing the effect of device heterogeneity by assigning different workload amounts to clients with different processing characteristics.

The resulting formulation is related to the Multiple-Choice Minimum-Cost Maximal Knapsack Packing Problem, in which each client defines a group of mutually exclusive choices corresponding to its feasible workload assignments (60, 57). Selecting one choice for a client determines both whether that client participates and how much workload it

receives. This perspective is appropriate for federated learning scenarios in which a fixed workload must be distributed among heterogeneous clients while optimizing execution time, energy consumption, or a combination of both.

2.4 Dynamic Programming

Dynamic Programming is an algorithmic technique for solving problems that can be decomposed into smaller, overlapping subproblems (61, 62, 57). Instead of repeatedly solving the same subproblem, dynamic programming stores its result and reuses it whenever that subproblem appears again. This principle can substantially reduce the computational cost compared with an exhaustive enumeration of all complete solutions.

2.4.1 State and Recurrence Formulation

A problem is particularly suitable for dynamic programming when it exhibits *optimal substructure*. This means that an optimal solution to the complete problem can be constructed from optimal solutions to appropriately defined subproblems (63, 62, 57). The technique is widely used in shortest-path algorithms, sequence alignment, knapsack problems, resource allocation, and scheduling problems involving discrete decisions.

A dynamic programming algorithm is usually described through four components: a state definition, a set of base cases, a recurrence relation, and a method for recovering the decisions that form the solution. A state summarizes all information from a partial solution that is necessary to extend it. The recurrence specifies how a state is obtained from smaller states, while the base cases initialize the smallest subproblems. When the actual decisions are required in addition to the optimal objective value, predecessor information is stored and later followed in a traceback phase.

Consider the workload-distribution problem defined in Section 2.3, in which the total workload t must be distributed among the eligible clients in $\mathcal{C}_r$. Let $n = |\mathcal{C}_r|$ denote the number of eligible clients in round r. Assume that these clients are considered in a fixed order, indexed from 1 to n. For each client $i \in \mathcal{C}_r$, the set $\widetilde{\mathtt{AC}}_{i,r}$ contains the feasible numbers of tasks that may be assigned to that client, including zero when the client may remain unselected.

A natural dynamic programming state is

$$Z_{i,r}(q), \tag{2.23}$$

where $Z_{i,r}(q)$ denotes the best objective value achievable after considering the first i clients in round r and assigning exactly q tasks. The state retains only the number of clients already considered and the amount of workload already assigned. Once the best objective value for a state has been stored, the individual decisions leading to that state do not need to be reconsidered (60, 57).

The base cases are

$$Z_{0,r}(0) = e, \qquad Z_{0,r}(q) = +\infty \quad \forall q > 0, \tag{2.24}$$

where e is the identity value of the objective aggregation operation. For an additive minimization objective, such as total energy consumption, $e = 0$. The value $+\infty$ denotes an infeasible state, since a positive workload cannot be assigned before any client has been considered.

Let $c_{i,r}(x_i)$ denote the cost of assigning x_i tasks to client i in round r. For an additive minimization objective, the recurrence is

$$Z_{i,r}(q) = \min_{\substack{x_i \in \widetilde{\mathtt{AC}}_{i,r} \\ x_i \leq q}} \{Z_{i-1,r}(q - x_i) + c_{i,r}(x_i)\} \tag{2.25}$$

Each transition considers one feasible assignment x_i for client i and combines it with the best partial schedule that assigns the remaining $q - x_i$ tasks to the first $i - 1$ clients. After all eligible clients have been considered, the optimal objective value for the complete workload is

$$Z_{n,r}(t) \tag{2.26}$$

For a bottleneck objective such as makespan minimization, the client costs are combined using the maximum operator. Let $c^{\text{time}}_{i,r}(x_i)$ denote the execution time of client i when it receives x_i tasks. The recurrence becomes

$$Z_{i,r}(q) = \min_{\substack{x_i \in \widetilde{\mathtt{AC}}_{i,r} \\ x_i \leq q}} \left\{\max\left(Z_{i-1,r}(q - x_i), c^{\text{time}}_{i,r}(x_i)\right)\right\} \tag{2.27}$$

In this recurrence, each candidate value is the largest execution time between the partial schedule formed by the first $i - 1$ clients and the execution time introduced by assigning x_i tasks to client i. Minimizing this value produces the workload allocation with the smallest makespan.

More generally, the recurrence may be expressed as

$$Z_{i,r}(q) = \underset{\substack{x_i \in \widetilde{\mathtt{AC}}_{i,r} \\ x_i \leq q}}{\text{opt}} \left\{ Z_{i-1,r}(q - x_i) \oplus c_{i,r}(x_i) \right\}, \tag{2.28}$$

where opt represents either minimization or maximization, and $\oplus$ is the objective-specific aggregation operator. For example, $\oplus$ may represent addition for total energy consumption, maximum for makespan, or a tuple-based comparison for lexicographic optimization.

2.4.2 Solution Recovery and Computational Complexity

To reconstruct the workload assignment, the algorithm stores, for each state, the assignment that produces its best value:

$$\widehat{x}_{i,r}(q) \in \underset{\substack{x_i \in \widetilde{\mathtt{AC}}_{i,r} \\ x_i \leq q}}{\arg\,\text{opt}} \left\{ Z_{i-1,r}(q - x_i) \oplus c_{i,r}(x_i) \right\} \tag{2.29}$$

Starting from the final state (n,t), the assignment to the last considered client is $\widehat{x}_{n,r}(t)$. The algorithm then moves to $(n-1, t-\widehat{x}_{n,r}(t))$ and repeats this process until the workload assignments of all clients have been recovered. The selected client set $\mathcal{S}_r$ is subsequently obtained from the reconstructed assignments according to Equation (2.21).

The recurrence may be evaluated either by memoization or by tabulation. Memoization follows a top-down approach and computes states on demand, whereas tabulation follows a bottom-up approach and evaluates states in an order that guarantees that all predecessor states are already available. When every state is potentially reachable, tabulation often provides a simpler implementation and more predictable memory access.

Let $A_{\max,r} = \max_{i \in \mathcal{C}_r} \left| \widetilde{\mathtt{AC}}_{i,r} \right|$ denote the largest number of feasible workload assignments available to any eligible client in round r. The recurrence evaluates at most $n(t+1)$ states and considers at most $A_{\max,r}$ transitions per state. Its time complexity is therefore $\mathcal{O}\left(n \cdot t \cdot A_{\max,r}\right)$, while storing the complete dynamic programming table requires $\mathcal{O}\left(n \cdot t\right)$ memory. If only the optimal objective value is required, the memory consumption can be reduced to $\mathcal{O}(t)$ by keeping only the current and previous table rows. Recovering the complete assignment, however, usually requires storing predecessor decisions or recomputing parts of the table.

This type of dynamic programming algorithm is described as pseudo-polynomial because its complexity depends on the numerical workload value t, rather than only on the

number of bits required to represent t. It can provide an exact solution for moderate workload sizes and finite assignment sets, but it may become impractical when the number of eligible clients, the total workload, or the number of feasible assignments per client becomes large. Additional state dimensions introduced for constraints such as energy budgets, deadlines, or minimum participation requirements may further increase both time and memory consumption.

Despite this limitation, dynamic programming is well suited to the scheduling formulation considered in this work. Each client $i \in \mathcal{C}_r$ provides a finite set of feasible workload choices $\widetilde{\mathtt{AC}}_{i,r}$, the total workload t is fixed, and a partial solution involving the first i clients can be summarized by the amount of workload q already assigned and the best objective value $Z_{i,r}(q)$ achieved. The resulting recurrence avoids enumerating every complete combination of client assignments and provides an exact baseline against which faster heuristics or approximate methods can be evaluated (60, 57).

2.5 Metaheuristics

Many optimization problems cannot be solved efficiently by exhaustive search or exact methods when the number of feasible solutions becomes very large. This is particularly common in combinatorial optimization, where the number of possible configurations grows rapidly with the problem size and each decision may affect the feasibility and quality of subsequent decisions. In such cases, metaheuristics provide a practical alternative for obtaining high-quality solutions within a limited computational budget (64, 65, 66).

2.5.1 Fundamentals and Classification

Metaheuristics are general-purpose search strategies that guide the exploration of a solution space without requiring its complete enumeration. Unlike exact algorithms, they generally do not guarantee that a globally optimal solution will be found. Instead, they attempt to identify sufficiently good solutions by repeatedly generating, modifying, and evaluating candidate solutions. This trade-off is useful when obtaining an exact solution would require excessive execution time or memory.

Let Ω denote the set of candidate solutions of an optimization problem, and let $\Omega_F \subseteq \Omega$ denote its feasible subset. For a minimization problem with objective function

$$f : \Omega_F \to \mathbb{R}, \tag{2.30}$$

the objective of a metaheuristic is to produce a solution $\widehat{s} \in \Omega_F$ with a low objective value:

$$f(\widehat{s}) \approx \min_{s \in \Omega_F} f(s) \tag{2.31}$$

The approximation in Equation (2.31) reflects the absence of an optimality guarantee. The quality of the resulting solution may be assessed by comparison with a known optimum, an exact method, a lower bound, or competing heuristic methods.

The design of a metaheuristic generally requires a solution representation, an initialization procedure, one or more mechanisms for generating candidate solutions, a rule for accepting or selecting candidates, and a stopping criterion (65, 66). The solution representation determines how the problem decisions are encoded. The initialization procedure constructs one or more starting solutions. Candidate-generation mechanisms define how new regions of the solution space are reached, while acceptance or selection rules determine which candidates influence the continuation of the search. The stopping criterion limits the computational effort, for example by specifying a maximum number of iterations, solution evaluations, or units of elapsed time.

A central aspect of metaheuristic design is the balance between *intensification* and *diversification*. Intensification concentrates the search around promising solutions or regions that have previously produced good objective values. Diversification directs the search toward different regions of the solution space and reduces the risk of repeatedly examining similar solutions or becoming trapped in a poor local optimum (64, 65, 66). Excessive intensification may cause premature convergence, whereas excessive diversification may prevent the method from sufficiently exploiting promising solutions.

Metaheuristics are commonly classified according to the number of candidate solutions manipulated during the search. From this perspective, they can be broadly divided into single-solution and population-based methods (65, 66). This classification describes the main search structure, although individual algorithms may combine characteristics from both categories.

Single-solution metaheuristics, also called trajectory-based methods, maintain one candidate solution and repeatedly transform it. Given a current solution $s \in \Omega_F$, a neighborhood function

$$\mathcal{N}(s) \subseteq \Omega \tag{2.32}$$

defines the candidate solutions that can be reached from s through one application of a prescribed modification. The algorithm generates a candidate $s' \in \mathcal{N}(s)$ and determines

whether it should replace the current solution according to an acceptance rule. The sequence of accepted solutions forms a trajectory through the solution space.

A basic local-search rule accepts a candidate when it improves the current solution:

$$s \leftarrow s' \quad \text{if} \quad f(s') < f(s) \tag{2.33}$$

However, accepting only improving candidates may cause the search to stop at a local optimum. A solution $s^\star$ is locally optimal with respect to $\mathcal{N}$ when

$$f(s^\star) \leq f(s'), \qquad \forall s' \in \mathcal{N}(s^\star) \tag{2.34}$$

A local optimum is not necessarily globally optimal because a better solution may exist outside the current neighborhood. Different single-solution metaheuristics employ different mechanisms for overcoming this limitation. Simulated Annealing may probabilistically accept worsening moves, Tabu Search uses memory structures to discourage cycling, Iterated Local Search perturbs locally optimal solutions, and Variable Neighborhood Search changes the neighborhood structure during the search (67, 68, 69, 70). Large Neighborhood Search modifies a larger portion of the candidate solution through destruction and reconstruction operations (71, 72).

Population-based metaheuristics, in contrast, maintain a collection

$$\mathcal{P} = \{s^{(1)}, s^{(2)}, \ldots, s^{(N_{\text{pop}})}\} \tag{2.35}$$

of N_{pop} candidate solutions. These methods update the population through mechanisms such as selection, recombination, mutation, cooperation, or information sharing. Genetic Algorithms produce new candidates using selection and genetic operators inspired by natural processes (73). Particle Swarm Optimization updates candidates according to individual and collective search information (74), while Ant Colony Optimization constructs solutions using artificial pheromone information accumulated during the search (75).

Maintaining multiple solutions can provide broad exploration and preserve information about different regions of the solution space. However, population-based methods may require a larger number of objective-function evaluations per iteration. Whether this cost is significant depends on the population size, the number of iterations, and the cost of evaluating each candidate. Population operations may also offer opportunities for parallel evaluation when sufficient computational resources are available.

In constrained optimization problems, candidate-generation mechanisms may produce

solutions outside Ω_F. Therefore, feasibility can be handled in several ways. A feasibility-preserving representation or operator generates only valid candidates. A repair procedure transforms an infeasible or incomplete candidate into a feasible one. Alternatively, constraint violations may be incorporated into the objective through a penalty function. For example, a candidate $s \in \Omega$ may be evaluated using $f_{\text{pen}}(s) = f(s) + \lambda \cdot V(s)$, where $V(s) \geq 0$ measures the degree of constraint violation and $\lambda > 0$ determines the corresponding penalty. Feasibility-preserving operators are particularly useful when repairing a candidate is difficult or when the objective function is meaningful only for feasible solutions.

2.5.2 Solution Representation and Neighborhood Design

Single-solution metaheuristics are especially suitable when the evaluation of candidate solutions is costly, when a good initial solution is available, or when problem-specific neighborhood operators can efficiently exploit the structure of the problem. In scheduling problems, a candidate solution may specify the resource assigned to each task, the order in which tasks are processed, or the amount of workload assigned to each resource. A neighboring solution may then be generated by moving a task or workload unit between resources, exchanging assignments, changing the workload of one or more resources, or replacing a selected resource with another.

For the workload-distribution formulation introduced in Section 2.3, a candidate solution can be represented by the assignment vector $\mathcal{X} = (x_i)_{i \in \mathcal{C}_r}$, where each x_i denotes the number of tasks assigned to client i. A feasible candidate must satisfy:

$$x_i \in \widetilde{\mathtt{AC}}_{i,r}, \qquad \forall i \in \mathcal{C}_r, \tag{2.36}$$

and

$$\sum_{i \in \mathcal{C}_r} x_i = t \tag{2.37}$$

As in Equation (2.20), a positive assignment simultaneously indicates that the corresponding client is selected. Consequently, a modification to $\mathcal{X}$ may alter both the selected-client set and the workload allocation.

A simple neighborhood could be defined by a workload-transfer move. Given two clients i and j, an amount δ may be removed from client i and assigned to client j:

$$x'_i = x_i - \delta, \qquad x'_j = x_j + \delta \tag{2.38}$$

All other assignments remain unchanged. The move preserves the total workload, but it is feasible only when

$$x_i - \delta \in \widetilde{\mathtt{AC}}_{i,r} \quad \text{and} \quad x_j + \delta \in \widetilde{\mathtt{AC}}_{j,r} \tag{2.39}$$

Such a move may reduce a bottleneck by transferring workload away from a slow or heavily loaded client. Nevertheless, changing only two assignments may be insufficient when reaching a better solution requires several coordinated changes.

2.5.3 Large Neighborhood Search

Large Neighborhood Search (LNS) is a single-solution metaheuristic that explores large neighborhoods by partially destroying and subsequently repairing a candidate solution (71, 72). Let s denote the current solution. A destruction operator D removes, relaxes, or modifies a subset of its decisions:

$$s^- = D(s), \tag{2.40}$$

where s^- is the destroyed solution. A repair operator R then reconstructs or adjusts the affected decisions:

$$s' = R(s^-), \tag{2.41}$$

producing a complete candidate solution s'. The resulting LNS neighborhood can therefore be interpreted as the set of candidates obtainable from the possible choices made by the destruction and repair procedures:

$$\mathcal{N}_{D,R}(s) = \{R(D(s))\} \tag{2.42}$$

Since the destruction and repair procedures may contain randomized decisions, repeated applications to the same current solution may produce different candidates.

The destruction operator determines which parts of the solution are initially allowed to change. It may remove decisions randomly to promote diversification or select related, costly, or otherwise undesirable decisions to intensify the search around a particular structural weakness. In a scheduling problem, destruction may target assignments associated with overloaded resources, slow resources, or a randomly selected subset of resources. The repair operator subsequently reconstructs the solution using a greedy rule, a randomized heuristic, a local-search procedure, a constraint-programming method, or an exact optimization method applied to the restricted subproblem (71, 72).

In the workload-allocation formulation in this work, the generic solutions s, s^-, and s' correspond to $\mathcal{X}_{\text{curr}}$, $\mathcal{X}_{\text{dest}}$, and $\mathcal{X}_{\text{rpr}}$, respectively. Let $\mathcal{X}_{\text{init}}$ denote the initial solution provided to LNS, and let $\mathcal{X}_{\text{curr}}$ denote the current solution maintained during the search. The destruction operator then selects a subset

$$\mathcal{C}_r^{\text{dest}} \subseteq \{i \in \mathcal{C}_r \mid x_i^{\text{curr}} > 0\} \tag{2.43}$$

of clients that are active in the current solution. In this work, the number of clients in $\mathcal{C}_r^{\text{dest}}$ is determined from a destruction sampling factor. For every client $i \in \mathcal{C}_r^{\text{dest}}$, a fraction of its current workload is removed, and the resulting assignment is adjusted to the largest feasible capacity that does not exceed the reduced value.

Let $\mathcal{X}_{\text{dest}}$ denote the destroyed solution. For every destroyed client,

$$x_i^{\text{dest}} \in \widetilde{\texttt{AC}}_{i,r}, \qquad x_i^{\text{dest}} \leq x_i^{\text{curr}}, \qquad \forall i \in \mathcal{C}_r^{\text{dest}}. \tag{2.44}$$

Clients that are not selected for destruction initially retain their current assignments:

$$x_i^{\text{dest}} = x_i^{\text{curr}}, \qquad \forall i \in \mathcal{C}_r \setminus \mathcal{C}_r^{\text{dest}}. \tag{2.45}$$

The workload removed by the destruction phase is $t_{\text{rem}} = t - \sum_{i \in \mathcal{C}_r} x_i^{\text{dest}}$. Since the current solution assigns exactly t tasks and the destruction operation only reduces client assignments, $t_{\text{rem}} \geq 0$ immediately after destruction.

The repair phase attempts to restore the total workload by moving through the discrete assignment capacities of the clients. Let

$$\text{prev}_{i,r}(x) = \max\{a \in \widetilde{\texttt{AC}}_{i,r} \mid a < x\} \tag{2.46}$$

denote the feasible capacity immediately below x, when such a value exists, and let

$$\text{next}_{i,r}(x) = \min\{a \in \widetilde{\texttt{AC}}_{i,r} \mid a > x\} \tag{2.47}$$

denote the feasible capacity immediately above x. During repair, an assignment can therefore be decreased by moving it to its previous feasible capacity or increased by moving it to its next feasible capacity.

Unlike a repair procedure restricted to the destroyed subset, the repair mechanism considered in this work may modify assignments associated with any client in $\mathcal{C}_r$. Consequently, the destroyed-client set identifies the initial perturbation, but it does not define the complete set of clients that may be changed during reconstruction. The preferred

clients are the non-destroyed ones, *i.e.*, selected clients whose assignments remain unchanged after the destroy phase and that still have additional assignment capacity. If no such client is available, the procedure may select a client with zero assigned tasks, thereby introducing a previously unselected client into the solution. As a fallback, any client with available capacity may also receive workload.

Let $\mathcal{X}_{\text{rpr}}$ denote the solution being repaired. At each repair step, its total assigned workload is $t_{\text{rpr}} = \sum_{i \in \mathcal{C}_r} x_i^{\text{rpr}}$. If $t_{\text{rpr}} < t$, the repair procedure selects a client with an available higher assignment capacity and applies $x_i^{\text{rpr}} \leftarrow \text{next}_{i,r}\,(x_i^{\text{rpr}})$.

Given that assignment capacities are discrete, increasing an assignment to its next feasible value may cause the total assigned workload to exceed t. Therefore, the repair procedure also handles the case $t_{\text{rpr}} > t$. In this case, a client with a lower feasible assignment capacity is selected and its assignment is updated according to $x_i^{\text{rpr}} \leftarrow \text{prev}_{i,r}\,(x_i^{\text{rpr}})$.

The repair procedure initially favors non-destroyed clients when removing excess workload, but it may use any client with removal capacity when necessary. It terminates successfully when $\sum_{i \in \mathcal{C}_r} x_i^{\text{rpr}} = t$. Moreover, every individual assignment must remain feasible: $x_i^{\text{rpr}} \in \widetilde{\texttt{AC}}_{i,r}, \forall i \in \mathcal{C}_r$.

The repair procedure is executed for a bounded number of internal iterations. If an exact repair is not obtained within this limit, additional workload-addition or workload-removal attempts are performed to reduce the difference between the assigned and required workloads. The candidate is nevertheless accepted only if the final workload and assignment-capacity constraints are satisfied.

The size and intensity of the destruction operation influence both the behavior and computational cost of LNS. Destroying a small number of clients or removing a small fraction of their workloads produces a neighborhood similar to conventional local search. It usually allows faster repair and stronger intensification but may be unable to escape a local optimum. Destroying a larger portion of the solution enables more substantial changes and promotes diversification, although it may also require more repair operations and generate candidates that differ considerably from the current solution.

After repair, the candidate solution is evaluated according to both feasibility and application-specific acceptance conditions. In this work, these conditions include the complete assignment of the workload, the validity of every client assignment, and an upper bound (deadline) τ on the predicted makespan of the repaired solution. In addition, percentage-change thresholds may be imposed on client diversity, makespan, and energy

consumption relative to the initial solution $\mathcal{X}_{\text{init}}$. These conditions restrict the search to candidates that satisfy the required performance and resource constraints.

When a repaired solution satisfies the acceptance predicate, it becomes the new current solution:

$$\mathcal{X}_{\text{curr}} \leftarrow \mathcal{X}_{\text{rpr}}. \tag{2.48}$$

Acceptance is therefore determined by the imposed constraints rather than solely by an improvement in a scalar objective. An accepted candidate may have a worse objective value than the best solution found so far, which allows the search trajectory to move through different feasible regions of the solution space.

The best-known solution is maintained separately from the current solution. Candidate solutions are compared through a weighted objective F that combines normalized scheduling and participation metrics.

After an accepted solution is evaluated, it replaces the best-known solution when

$$F\left(\mathcal{X}_{\text{rpr}}\right) < F\left(\mathcal{X}_{\text{best}}\right). \tag{2.49}$$

In this case,

$$\mathcal{X}_{\text{best}} \leftarrow \mathcal{X}_{\text{rpr}}. \tag{2.50}$$

Separating the current solution from the best-known solution allows the method to accept feasible transitions without losing the best result already obtained.

The LNS execution continues until a stopping criterion is satisfied. Common stopping criteria include a maximum number of iterations and a maximum elapsed time. Iteration-based termination provides a fixed number of destruction-and-repair attempts, whereas time-based termination provides a direct computational budget. The latter may be useful when scheduling decisions must be produced within a limited interval.

The computational cost of a metaheuristic depends on the number of iterations, the number of generated candidates, and the cost of constructing and evaluating each candidate. If N_{it} iterations are performed, at most N_{cand} candidates are evaluated per iteration, and each candidate requires $\mathcal{O}(E)$ time to construct and evaluate, the generic computational cost can be expressed as $\mathcal{O}\left(N_{\text{it}} \cdot N_{\text{cand}} \cdot E\right)$.

For the single-solution LNS procedure considered here, one repaired candidate is generated per outer iteration, and therefore $N_{\text{cand}} = 1$. The value of E depends on the destruction operation, the number of internal repair steps, the evaluation of the schedul-

ing metrics, and the validation of the acceptance conditions.

Let I_{rpr} denote the maximum number of internal repair iterations. Since each repair step may inspect the set of eligible clients to identify valid candidates, the repair procedure may require up to $\mathcal{O}(I_{\text{rpr}} \cdot |\mathcal{C}_r|)$ operations in the worst case, excluding the cost of evaluating the repaired solution. The complete LNS execution may therefore be expressed generically as $\mathcal{O}(N_{\text{it}} \cdot (I_{\text{rpr}} \cdot |\mathcal{C}_r| + E_{\text{eval}}))$, where E_{eval} denotes the cost of estimating and normalizing the scheduling metrics and verifying the acceptance conditions. Unlike the pseudo-polynomial DP complexity derived in Section 2.4, this expression represents a configurable search budget rather than the cost required to prove optimality.

Metaheuristics are suitable for the scheduling formulation considered in this work because the complete set of client-assignment combinations may become too large for exact enumeration, particularly when many clients and feasible assignment capacities are available. A feasible workload allocation can be directly represented by the vector $\mathcal{X}$, and the destruction-and-repair operators can modify client selection and workload allocation jointly. The destruction phase perturbs the assignments of a subset of active clients, while the global repair phase may reallocate workload among the complete set of eligible clients and may include previously unselected clients.

3 Literature Review

Client selection is a central component of Federated Learning (FL) that directly affects convergence speed, resource efficiency, model quality, and participation fairness. In cross-device FL, this problem is particularly challenging because clients may differ substantially in terms of data distribution, computation capacity, energy availability, network quality, and availability over time (14, 16, 13, 76). Consequently, selecting clients based only on a single criterion, such as data size, training speed, or update quality, may lead to inefficient training, biased participation, or degraded generalization under heterogeneous conditions.

Existing client selection methods address different subsets of these challenges. Some approaches focus on statistical heterogeneity and convergence, while others focus on system efficiency, energy consumption, or communication constraints. Another group explicitly targets fairness or diversity. More recent works also explore optimal and metaheuristic formulations to handle the combinatorial nature of the selection problem. These categories are not exclusive, as several methods overlap across different challenges and optimization goals. However, most existing methods still treat client selection as a binary decision, where a client is either selected or not selected, without explicitly controlling how much workload each selected client processes. Moreover, few approaches jointly consider system-level efficiency, energy consumption, participation diversity, class-distribution quality, reliability, and learning behavior through historical loss information.

This chapter reviews the main categories of client selection approaches related to the scope of this thesis, using a concise grouping of dominant perspectives rather than a strict taxonomy. Section 3.1 discusses methods addressing statistical heterogeneity, model convergence, and data-sampling behavior. Section 3.2 reviews resource-, energy-, carbon-, and communication-aware approaches. Section 3.3 examines works that address dynamic client availability. Section 3.4 examines methods focused on fairness, diversity, bias, explainability, and trustworthy selection. Section 3.5 discusses optimal scheduling-based methods. Section 3.6 reviews metaheuristic-based approaches. Finally, Section 3.7 summarizes the positioning of this work with respect to the literature, while Section 3.8 consolidates the target environment, design choices, and scope boundaries.

3.1 Statistical and Utility-Aware Methods

In real-world FL, clients often hold non-IID data, which can lead to divergent local updates, slower global convergence, and differences in client informativeness across training stages (77). Since the standard FedAvg algorithm relies on random client sampling, it does not explicitly identify which clients provide useful data, gradients, or learning signals in a given round. Several methods therefore adapt the selection process to prioritize clients according to data representativeness, update quality, contribution estimates, or learning-phase information. These approaches are mainly concerned with improving convergence and model utility under statistical heterogeneity. Here, utility denotes a method-specific scalar proxy for the learning benefit of selecting a client, which may be derived from loss values, gradient alignment, data quality, validation performance, or historical contribution. It is not used in the formal sense of an α-fair network-utility function, and it should not be interpreted as a direct measure of participation or resource-allocation fairness.

FedAvg (1) adopts random client selection as its baseline strategy. At each communication round, the server samples a fraction C of the available clients, sends them the current global model, and aggregates the returned local models by weighted averaging according to the number of local examples. The method improves communication efficiency mainly by allowing selected clients to perform multiple local updates before aggregation, controlled by the number of local epochs E and minibatch size B. Thus, client selection in FedAvg is not resource-aware or utility-aware; it is primarily a random sampling mechanism used to enable scalable synchronous FL.

FedSampling (78) addresses the bias caused by uniform client sampling when clients hold highly imbalanced amounts of data. Instead of sampling clients uniformly or weighting clients directly by local dataset size, it samples local data with an identical probability across all available clients. In each round, the server defines a desired number of samples K and estimates the total number of samples N among available clients through a local differential privacy mechanism, since exact local dataset sizes may be privacy-sensitive. Each client then independently samples its local examples with probability $K/\widetilde{N}$ and builds its local update from the sampled data. Thus, FedSampling tackles client selection indirectly, by replacing client-level sampling with privacy-preserving data-level sampling to reduce biased data exploitation under imbalanced client data sizes.

Favor (79) addresses client selection under non-IID data by using reinforcement learning to actively choose the participating devices in each round. The method relies on the

observation that local model weights carry information about the underlying data distribution of each client. Thus, instead of directly accessing local data, Favor represents the selection state using the global model and the latest local model weights, compressed through Principal Component Analysis (PCA) when needed. A Double Deep Q-Network (DQN) agent then estimates the expected benefit of selecting each client and chooses the top-K devices for training. Its reward encourages higher validation accuracy while penalizing additional communication rounds, so the selection policy is learned to reduce the number of rounds required for convergence.

FedPNS (9) addresses client selection under non-IID data through a probabilistic node selection mechanism. The method first applies an *Optimal Aggregation* procedure to the clients selected in a round, identifying local updates that may harm convergence by analyzing the relationship between each local gradient and the global gradient. Updates considered adverse can be excluded from aggregation. The result of this procedure is then used to adjust each client's future selection probability: clients whose updates are frequently identified as adverse receive lower probabilities, while the remaining clients become more likely to be selected. Thus, FedPNS tackles client selection by learning contribution-aware sampling probabilities over rounds, favoring clients whose updates are better aligned with global convergence.

Other approaches incorporate client contributions or quality-related signals into the selection process. *AUCTION* (80) formulates client selection as a budget-constrained decision problem in an FL services market, where candidate clients differ in data size, data quality, and claimed price. It uses a reinforcement learning policy network based on an attention-based encoder-decoder architecture to sequentially select clients within the available budget. The client state includes the number of local samples, label-quality and distribution-quality indicators, and the required payment, while the reward is derived from the FL model performance obtained after selecting a client subset. *AdaFL* (81) adapts client selection by dynamically changing the number of selected clients over training rounds and by evaluating clients through both current and historical contribution estimates. Its contribution score combines local dataset size and loss information, and then smooths this score across rounds using a weighted average with a decay mechanism. These methods highlight the role of quality-aware, contribution-aware, and history-aware criteria when estimating client usefulness.

Oort (46) guides participant selection by jointly considering statistical utility and system efficiency. For training, it estimates each client's statistical utility from the aggre-

gate training loss observed in previous rounds, using higher loss as an indication that the client's data may still provide useful learning signal. This utility is combined with system information, such as the expected training duration of the client, so that slow clients can be penalized when they are likely to delay aggregation. Oort then applies an online exploration-exploitation strategy: it prioritizes clients with high estimated utility while still exploring previously unselected clients to handle stale or incomplete information.

CriticalFL (82) adapts client selection by exploiting the notion of critical learning periods, *i.e.*, early training phases in which insufficient or low-quality learning signals can permanently degrade the final model. Instead of selecting a fixed number of clients throughout training, CriticalFL detects these periods in an online manner using the Federated Gradient Norm (FGN), which tracks changes in the aggregate training-loss variation across selected clients. When a critical period is detected, the method increases the number of selected clients to expose the model to more data during the most sensitive phase of training. After this period, it gradually reduces the number of selected clients to improve communication efficiency. Thus, CriticalFL tackles client selection mainly by adapting the selection size over time according to the current learning phase.

Overall, statistical and utility-aware methods improve client selection by prioritizing data representativeness, gradient alignment, contribution estimates, or learning-phase behavior. However, their main decision is typically whether a client should participate, not how much work that client should process once selected. Moreover, these methods usually do not combine learning-aware utility with execution time, energy consumption, participation diversity, reliability, and explicit class-distribution control in the same scheduling formulation. In contrast, MetaCS-FL incorporates historical training and testing losses as part of a broader multi-objective client-task scheduling problem, allowing learning behavior to influence selection together with system efficiency, fairness, data-distribution quality, and workload assignment. More specifically, the MetaCS-FL utility score is a workload-weighted combination of training- and testing-derived historical performance terms. The parameter α in this combination is only a convex mixing coefficient controlling the relative importance of these two terms and is unrelated to α-fairness.

3.2 System- and Communication-Aware Methods

Client participation in FL is often constrained by computation, communication, energy, and sustainability limitations. In heterogeneous cross-device and wireless environments,

selecting clients without considering these constraints can increase round duration, drain client batteries, lead to missed deadlines, intensify wireless interference, or raise the carbon cost of training. Resource-aware methods, therefore, aim to avoid slow or energy-expensive participants, improve communication efficiency, respect energy or deadline constraints, reduce straggler effects, and allocate wireless or compute resources more effectively. These approaches view client selection as a system optimization problem driven by latency, energy, wireless conditions, resource allocation, carbon emissions, or resource availability.

FedCS (7) addresses client selection in mobile edge environments where clients differ in computation capacity, wireless channel quality, and local data size. Before each round, the server sends a resource request to a random subset of available clients and collects information about their resource conditions. Based on this information, FedCS estimates the time required for model distribution, local training, and upload, and then selects clients that can complete the round before a predefined deadline. The selection objective is to maximize the number of client updates aggregated within the deadline, using a greedy scheduling procedure that prioritizes clients with lower expected update and upload times. Thus, FedCS tackles client selection as a resource-aware scheduling problem to reduce round duration and improve training efficiency under heterogeneous system conditions.

Ouyang and Liu (83) address client selection in wireless FL by combining contribution-aware participation with batch-size control. The method defines each client's contribution from the projection of its local gradient onto the global gradient, capturing both gradient direction and magnitude. In each round, the server computes a contribution threshold and batch size, and clients only upload their local gradients when their estimated contribution exceeds this threshold. The threshold is not fixed empirically; it is optimized jointly with the local batch size to maximize learning efficiency, defined as the expected convergence improvement per unit of learning time. Thus, the approach tackles client selection as a dynamic contribution-thresholding problem that also accounts for wireless delay, computation time, and client energy constraints.

FedMCCS (5) addresses client selection in IoT-based FL by considering multiple device and data-related criteria. The method first applies stratified sampling to filter clients according to their location metadata, aiming to reduce unnecessary communication with clients that are unlikely to be available in the same time zone. Then, among the filtered clients, it requests resource information and uses a linear-regression-based predictor to estimate whether each client has enough CPU, memory, energy, and time to complete the local training and upload steps. FedMCCS also considers the event rate of each client's

local data to prioritize clients that better represent minority-class samples in imbalanced classification tasks. Thus, it tackles client selection as a multicriteria optimization problem that seeks to maximize the number of reliable clients while reducing discarded rounds and improving training efficiency.

GREED (6) addresses client selection in green FL, where IoT clients rely on harvested energy and battery reserves. Unlike deadline-based methods that only consider whether clients can finish training and uploading on time, GREED also checks whether each selected client has enough available energy to complete the round. The method formulates selection as a trade-off between maximizing the number of selected clients and minimizing the energy drawn from their batteries. It first filters clients whose energy and deadline constraints can be satisfied, then prioritizes clients according to a score that combines their expected computation and upload energy with the selection-energy trade-off parameter. Thus, GREED tackles client selection as an energy-aware scheduling problem that preserves client battery lifetime while ensuring that selected clients can upload their local models before the deadline.

EACS-FL (84) addresses client selection under heterogeneous long-term energy constraints. The method formulates client selection as an online problem, since the server does not know client delay and energy consumption before selecting clients. It uses a Combinatorial Multi-Armed Bandit (CMAB) strategy to estimate each client's quality from previous observations of training delay, model download time, model upload time, and energy consumption. To prevent overusing clients, EACS-FL introduces a virtual energy deficit queue for each client and applies Lyapunov optimization to penalize clients whose energy consumption exceeds their allocated share of the long-term budget. Thus, EACS-FL tackles client selection as an online energy-aware optimization problem that balances delay reduction, energy-budget compliance, and selection frequency over time.

ESCS (85) addresses client selection from an energy-saving perspective while preserving model performance. The method first filters available clients according to minimum resource requirements, checking whether the expected battery level after training remains above a threshold and whether network quality is sufficient. Among the remaining clients, ESCS computes a utility score using battery level, model training energy consumption, training time, network quality, and evaluation loss. The method supports both system-based and model-based utility functions, as well as probabilistic or deterministic selection, allowing the selection behavior to be adapted to different application scenarios. Thus, ESCS tackles client selection as a configurable resource-aware mechanism that prioritizes

energy preservation while maintaining useful client contributions to model training.

FedAECS (12) addresses client selection in mobile edge computing by balancing energy consumption and learning accuracy. The method formulates selection as a nonlinear integer optimization problem that minimizes the ratio between total client energy consumption and the expected FL accuracy, while considering constraints on training time, bandwidth, transmission power, CPU frequency, and minimum accuracy. Since the problem is treated as a multidimensional knapsack problem, FedAECS uses a heuristic procedure that prioritizes clients according to learning time, local data size, channel quality, and the energy-accuracy ratio. It then recursively explores the prioritized list to select clients that satisfy the accuracy and resource constraints. Thus, FedAECS tackles client selection as an energy-accuracy trade-off problem for choosing clients that contribute sufficient data while limiting the energy cost of training and transmission.

FEPCS (86) addresses client selection in mobile edge FL through fine-grained energy planning. The method combines client-level and system-level mechanisms: it adapts each client's local workload according to residual energy, estimates transmission energy from historical channel information, and progressively increases the participation rate over training rounds. FEPCS also tracks an energy budget deviation metric to avoid overusing clients that consume energy faster than their budget allows. Client selection is guided by a unified score that combines energy cost, budget deviation, and data contribution, favoring clients that provide greater utility relative to their energy cost. Thus, FEPCS tackles client selection as a long-term energy-planning problem that aims to reduce dropouts, preserve client availability, and maintain useful data participation throughout training.

Fed-RAC (87) addresses participant heterogeneity in FL through resource-aware clustering rather than direct per-round client selection. The method first collects each participant's processing speed, data transmission rate, and available memory, normalizes these resource vectors, and uses Dunn indices to determine the number of clusters. Participants are then assigned to clusters according to their ability to train the corresponding model within a maximum allowable response time, while also satisfying precision and optimization-error thresholds. Each cluster uses a model size adapted to its resource capacity, and lower-resource clusters are improved through a leader-follower knowledge-distillation mechanism from the highest-resource cluster. Thus, Fed-RAC tackles client participation by grouping heterogeneous clients into resource-compatible clusters, reducing straggler effects while preserving the contribution of low-resource participants.

ELASTIC (10) addresses client selection in wireless FL by jointly considering la-

tency, energy consumption, and resource allocation. The method argues that maximizing the number of selected clients is not always desirable, as selecting fewer clients in early rounds and more clients in later rounds can reduce energy consumption while preserving learning performance. ELASTIC formulates selection as a trade-off between maximizing the number of clients that can finish before a deadline and minimizing their total energy consumption. It explicitly models computation latency, upload latency, and waiting time under time-division wireless access, and optimizes CPU frequency and transmission power for selected clients. Thus, ELASTIC tackles client selection as a joint scheduling and resource-management problem that balances participation size and energy cost under deadline constraints.

FedAEB (88) addresses client selection in heterogeneous wireless FL by jointly considering model performance, energy consumption, and latency. The method introduces a Data Quality Factor (DF) to indirectly capture the usefulness of each client's data distribution without requiring clients to reveal their local distributions. In each round, the server observes client states, including data quality, data size, channel conditions, bandwidth, and remaining energy, and formulates selection as a Markov Decision Process (MDP). The action space includes both client selection and transmission power allocation. FedAEB then uses a Soft Actor-Critic (SAC) deep reinforcement learning algorithm to learn a policy that maximizes data quality while penalizing latency and energy consumption under resource constraints. Thus, FedAEB tackles client selection as an online learning-based resource allocation problem that balances accuracy, energy, and latency in time-varying heterogeneous FL systems.

RaFed (89) addresses latency-aware FL in Industrial Internet of Things (IIoT) environments, where densely deployed devices may suffer from wireless interference when many clients transmit updates simultaneously. The method observes that selecting more clients can reduce the number of FL iterations but may also increase packet collisions and communication latency. RaFed therefore formulates client selection and wireless channel allocation as an optimization problem that minimizes the total training latency, considering both local computation time and model-upload time under packet reception constraints. Since the problem is NP-hard, it applies a heuristic procedure that iteratively adjusts the active devices and their wireless resources to reduce the longest device latency. Thus, RaFed tackles client selection as a latency-aware resource allocation problem that balances convergence speed and wireless interference in IIoT-based FL.

FedSynthesis (52) addresses client selection from a carbon-aware FL perspective. The

framework extends Flower by integrating CodeCarbon to measure client-side carbon emissions during local training and MLFlow to track metrics and results. After each round, clients report their model parameters together with measured CO_2-equivalent emissions, which are stored by the server and used to estimate each client's emissions for the next round. FedSynthesis provides three estimation strategies: simple averaging, exponential smoothing, and linear regression. The estimated emissions are then converted into selection probabilities, giving higher priority to clients expected to produce lower carbon emissions. Thus, FedSynthesis tackles client selection as a measurement-based carbon-aware sampling problem that prioritizes lower-emission clients while preserving the standard FL aggregation workflow.

Dynamic and volatile wireless scenarios have also been studied in over-the-air FL. *DCSE* (90) addresses client selection when clients have limited energy and may drop out during local training. The method analyzes how volatile clients affect aggregation quality, convergence speed, and wasted energy, showing that training performance depends on the effective aggregation size and the power scaling factor. Since the server does not know each client's local training state in advance, DCSE formulates selection as an online decision problem and applies an Exp3-based strategy with multiple plays and energy constraints to reduce wasted energy while improving convergence. *DOCS* (91) focuses on over-the-air FL with analog gradient aggregation, where channel noise and aggregation error affect the usefulness of selected clients. It transforms the global optimization problem into a client selection problem under a mean-squared-error constraint and dynamically removes clients that degrade the trade-off between aggregation size and power scaling. Thus, both methods tackle client selection in over-the-air FL by accounting for wireless aggregation effects, but DCSE emphasizes volatility, dropout, and energy waste, whereas DOCS emphasizes aggregation-error-aware client selection under channel noise.

FedJoule (92) addresses client selection under a global energy budget for FL over heterogeneous edge accelerators. The method jointly considers which clients participate and which device power modes they use, aiming to maximize model accuracy while keeping the total energy consumed across rounds within a predefined budget. FedJoule decomposes the problem into two Integer Linear Programming (ILP) subproblems: one for client selection and another for power-mode assignment. Client selection is guided by approximate Shapley values, which estimate each client's expected contribution to global accuracy, together with a cooldown mechanism to avoid repeatedly selecting the same clients. Power modes are chosen from an energy-time Pareto front built from profiling and prediction models, reducing energy without increasing the round duration. Thus,

FedJoule tackles client selection as a global energy-budgeted optimization problem that combines contribution-aware selection with accelerator-level power planning.

Overall, system-aware methods improve FL efficiency by considering latency, compute speed, communication quality, wireless interference, energy budgets, carbon emissions, power modes, resource allocation, or wireless dynamics. However, most of them optimize a limited system-level trade-off, such as time-energy, energy-accuracy, latency-interference, or carbon-aware participation, and only some provide workload control. Even when methods jointly optimize client selection with wireless resources, transmission power, CPU frequency, or device power modes, they usually do not integrate fairness, data-distribution quality, reliability-aware task capacities, and historical loss-based utility into the same decision process. MetaCS-FL builds on this line of work by jointly considering makespan and energy while also assigning variable workloads, encouraging diverse participation, accounting for class coverage and imbalance through privacy-preserving histograms, and limiting the workload of clients with weaker availability or completion history.

3.3 Dynamic Availability-Aware Methods

Beyond resource and data heterogeneity, cross-device FL must handle dynamic client availability. In practical deployments, clients may join after training has started, disconnect temporarily due to network instability or battery depletion, become unavailable during a round, or return after periods of inactivity. These dynamics affect both convergence and system efficiency because the set of eligible clients changes over time and because unreliable clients may fail to complete their assigned work. Availability-aware methods therefore try to model, predict, compensate for, or adapt to changing participation patterns.

AEDFL (93) addresses FL without a central server by adopting an asynchronous decentralized architecture for heterogeneous devices. Devices communicate only with neighbors in an exponential topology, while a lightweight coordinator manages device indices and heartbeats without participating in training or aggregation. Each device updates its local model, caches models received from neighbors, and dynamically decides which neighbor models should be aggregated. AEDFL uses a reinforcement-learning-based model selection mechanism to avoid unhelpful or stale neighbor models, and a dynamic weight update strategy that assigns larger weights to fresher and more useful models. It also applies adaptive sparse training to reduce computation and communication costs. Thus, AEDFL tackles client participation in decentralized FL by replacing server-side

client selection with asynchronous neighbor-model selection, staleness-aware aggregation, and resource-efficient sparse updates.

FedCAB (17) addresses client selection under availability budgets, where clients may have limited communication opportunities or may join the FL process only in later rounds. The method ranks available clients using two main signals: the Kullback-Leibler (KL) divergence between the local update and the global model, and the client's historical participation frequency. In early rounds, FedCAB uses a decaying quadratic function to give higher priority to clients with larger local-global distribution differences, encouraging the global model to learn from statistically distinctive clients. As training progresses, this preference fades and the ranking shifts toward clients with lower divergence to support convergence. FedCAB also compensates late-joining or less-active clients through additional ranking weights, increasing their chance of being selected before their communication budget is exhausted. Thus, FedCAB tackles client selection as an availability-aware probabilistic ranking problem that balances data heterogeneity, late participation, and limited communication budgets.

CA-Fed (18) addresses FL under heterogeneous and correlated client availability. The method is motivated by the fact that clients may be available with different frequencies and that their availability can be temporally or spatially correlated, for example, due to charging patterns, mobility, or time-zone effects. CA-Fed analyzes how such correlation affects convergence and shows a trade-off between faster convergence and bias in the learned model. Based on this analysis, the server dynamically estimates each client's availability, correlation level, and local loss, and then adjusts aggregation weights during training. Clients with low availability or high temporal correlation can be ignored when their participation is expected to slow convergence or increase instability. Thus, CA-Fed tackles client selection as a correlation-aware availability problem that balances convergence speed, aggregation bias, and the stability of client participation.

TwoStageEAFL (19) addresses client selection in dynamic wireless FL networks where clients may be unavailable due to mobility, limited battery supply, or competing local workloads. The method jointly optimizes client selection and communication resource allocation to balance model accuracy and energy consumption under a training-time constraint. To reduce repeated decision-making overhead, it decomposes the problem into two asynchronous stages. In the first stage, the server estimates each client's online probability from historical participation and preselects more stable long-term clients with their resource blocks. In the second stage, during each practical training epoch, tem-

porary clients are added as backups when some long-term clients are unavailable, and resources are adjusted accordingly. Client selection is handled through a greedy energy-accuracy-balanced procedure, while resource allocation uses importance sampling. Thus, TwoStageEAFL tackles client selection as a stagewise energy-accuracy optimization problem that combines long-term availability prediction with real-time backup selection.

FedAWE (20) addresses FL under heterogeneous and non-stationary client unavailability. The method shows that FedAvg can become biased when some clients are available more often than others or when availability changes over time. Instead of requiring the server to know client availability probabilities or store gradients for unavailable clients, FedAWE modifies the update process with two mechanisms. First, adaptive innovation echoing scales a client's local update according to the number of rounds since it was last available, helping compensate for missed computation. Second, implicit gossiping delays the global-model broadcast so that active clients indirectly mix their local updates through the server, promoting a more balanced diffusion of information. Thus, FedAWE tackles client unavailability as a bias-correction problem, using lightweight update reweighting and implicit information mixing without client selection or availability prediction.

FLICS-OPT (21) addresses client selection in large-scale FL with intermittent clients, time-varying availability, and communication constraints. The method assumes that clients are organized into groups by their data distributions since tracking individual clients is impractical when each client may participate only once. At each round, the server observes the number of available clients per group and the communication budget, solving an optimization problem to decide how many clients should be sampled from each group. This decision aims to achieve long-term per-group participation while minimizing the variance introduced by client sampling. Selected clients participate probabilistically within their groups, and the server uses importance sampling during aggregation to reduce bias against the target group weights. Thus, FLICS-OPT tackles client selection as a group-level sampling problem that adapts to intermittency and communication limits while preserving balanced participation across heterogeneous data groups.

RIFLES (22) addresses client selection by treating FL participation as a scheduling problem over multiple rounds. Instead of selecting clients from past information, the method builds an availability forecasting layer from client heartbeat messages, which indicate whether a device is connected, charged, and idle. These heartbeat traces are organized into daily availability matrices and processed with a CNN-LSTM model to predict client availability. Based on the predicted availability and each client's expected

response duration, RIFLES constructs an eligibility matrix indicating which clients are likely to remain available long enough to complete training. The framework then schedules clients using adaptive policies, including a greedy heuristic that prioritizes eligible slots with many clients and promotes unique-client participation, and an LRU (Least Recently Used) policy that favors clients that have not participated recently. Thus, RIFLES tackles client selection as a long-term availability-aware scheduling problem that combines forecasting, response-time estimation, historical participation, and adaptive round timing.

Overall, dynamic-availability methods show that client intermittency, late participation, correlated availability, non-stationary participation, and asynchronous or decentralized coordination can strongly affect FL training. However, these works usually focus on availability prediction, aggregation adjustment, probabilistic sampling, or backup selection, rather than on fine-grained server-side workload scheduling. MetaCS-FL addresses availability from a scheduling perspective: initially available clients are profiled before training, late-joining clients are incorporated after asynchronous profiling, returning clients can reuse stored profiles, and reliability-aware capacity sets reduce the workload assigned to clients with poor availability or completion history. This allows the framework to adapt both the selected clients and their task assignments as the candidate set evolves.

3.4 Fairness- and Diversity-Aware Methods

Unfair or biased client selection can repeatedly favor high-performing, resource-rich, or frequently available clients, reducing participation opportunities for others and potentially weakening model generalization. Such bias may arise from non-IID data distributions, client sampling, unequal participation histories, heterogeneous resource conditions, and differences between local and global objectives (94). Fairness-aware and diversity-aware approaches attempt to mitigate these effects by modifying aggregation weights, encouraging representative updates, promoting broader participation, or making client contributions more transparent.

q-FedAvg (95) addresses fairness in FL by changing how client updates are weighted during aggregation. The method is designed to solve the q-Fair Federated Learning (q-FFL) objective, which assigns higher importance to clients with larger local losses, so that poorly performing clients receive more influence in the global update. The parameter q controls the fairness-accuracy trade-off: $q = 0$ recovers the standard FedAvg objective, while larger q values increasingly prioritize clients with worse performance. In each round,

selected clients perform local SGD as in FedAvg, but their updates are rescaled using their local loss and an estimate of the local Lipschitz constant. Thus, q-FedAvg tackles client contribution as a fairness-aware aggregation problem that improves accuracy uniformity across clients without explicitly changing the client selection rule.

ECSM (96) addresses client selection by combining multiple criteria instead of relying on a single selection signal. The method ensembles three mechanisms: accuracy-based selection, reputation-based selection, and random selection. Accuracy-based selection prioritizes clients whose local updates achieve higher evaluation accuracy on the server side. Reputation-based selection measures the consistency of each client's accuracy across communication rounds, giving priority to clients with more reliable past performance. Random selection is retained as part of the mechanism to avoid repeatedly selecting only high-ranked clients and to improve model generalization across participants. Thus, ECSM tackles client selection as a multi-criteria ranking problem that balances client performance, historical reliability, and exploration during FL training.

DivFL (15) addresses client selection by promoting diversity among selected client updates. The method observes that many clients may provide redundant gradient information, so aggregating all or randomly selected updates can waste communication and reduce representativeness. In each round, DivFL selects a small subset of clients whose aggregated gradients approximate the aggregation that would be obtained from all clients. To achieve this, it defines client similarity in the gradient space and maximizes a submodular facility-location function using a greedy algorithm to choose representative clients under a cardinality constraint. The selected clients then perform local training as in FedAvg, and the server aggregates only their updates. Thus, DivFL tackles client selection as a diversity-aware subset selection problem that reduces redundant communication while improving learning efficiency and client-level fairness.

FedTune-SGM (97) addresses FL under data heterogeneity and resource-constrained edge devices by combining personalized fine-tuning, accuracy-based aggregation, and incentive control. The method first trains a base model at the cloud server and distributes its parameters to edge devices. Each device freezes the shared layers, adds personalized layers according to its computational capacity, and fine-tunes the model on local data to capture device-specific features. After local training, devices report their updated parameters and accuracy improvement, and the server aggregates models using accuracy-based weights so that higher-performing devices have stronger influence. FedTune-SGM also applies a Stackelberg Game Model (SGM), where the server acts as the leader and re-

wards devices according to accuracy improvement, while devices balance training effort against computational cost. Thus, FedTune-SGM tackles client contribution as a personalized and incentive-aware aggregation problem that promotes accurate participation under heterogeneous data and resource constraints.

xCS (98) addresses client selection by making it effective, explainable, and trustworthy. The method models client updates as players in a cooperative game and estimates each client's contribution to the global model using game-theoretic indices. Besides Shapley Values, xCS considers Banzhaf, Least Core, and Johnston indices as lighter alternatives for computing client contributions. These contribution values are used by two explicit selection policies: a ranking policy, which selects the top-contributing clients, and a quartile policy, which prioritizes clients in the highest contribution quartile. Thus, xCS tackles client selection as an explainable contribution-ranking problem that selects high-impact clients while providing auditable reasons for their participation.

Overall, fairness- and diversity-aware methods reduce selection bias, promote broader participation, improve robustness to heterogeneous contributions, and make selection decisions more explainable. However, they usually address fairness through aggregation rules, contribution scores, update diversity, or incentive mechanisms, without explicitly balancing fairness against execution time, energy consumption, data-distribution quality, and workload feasibility. MetaCS-FL treats fairness as part of the scheduling objective itself by using a recent-participation diversity score, so the framework can trade participation diversity against makespan, energy, class-distribution quality, reliability-aware capacities, and historical loss-based utility when deciding both which clients participate and how much data they process.

3.5 Optimal Scheduling-Based Methods

Some works formulate client selection or workload assignment as optimization problems with provable properties for specific objectives. These methods are especially relevant because they move beyond simple binary client selection and explicitly determine how much data (*i.e.*, how many tasks) each selected device should process. In this group, the central goal is typically to optimize one or two system-level metrics, such as minimizing the round makespan, minimizing total energy, or satisfying a deadline while assigning workloads across heterogeneous clients.

Fed-LBAP (99) addresses client scheduling under computational heterogeneity by

using workload size as a tuning knob. For IID data, the method profiles each mobile device to estimate training time as a function of the assigned data size and then formulates scheduling as a min-max problem that minimizes the slowest client's completion time. It combines data partitioning with a Linear Bottleneck Assignment Problem (LBAP), assigning more data to faster devices and less data to slower ones while preserving the required total workload. For non-IID data, the related *MinCost* method adds an accuracy cost based on each client's class distribution, penalizing harmful outliers while preserving clients that contain missing classes. Thus, Fed-LBAP and MinCost tackle client selection as workload-aware scheduling problems that balance round duration, device heterogeneity, and data-distribution effects.

OLAR (8) addresses FL round-duration reduction by controlling how many training tasks are assigned to each heterogeneous device. The method formulates the problem as makespan minimization with identical and independent tasks, where each device has a non-decreasing cost function and lower and upper limits on the number of tasks it can process. OLAR starts from the lower task limits and iteratively assigns each task to the device whose next task produces the smallest cost, using a minimum heap to keep this decision efficient. Unlike heuristic workload-assignment methods, OLAR proves that this greedy incremental assignment yields an optimal schedule under the stated assumptions. Thus, OLAR tackles client participation as an optimal workload-assignment problem that reduces straggler impact while respecting device-specific task limits.

Similarly, *(MC)²MKP* (60) addresses FL energy minimization by controlling how many training tasks each heterogeneous device processes. The method formulates workload assignment as the Minimal Cost FL Schedule problem, where devices have arbitrary energy-cost functions and lower and upper task limits. This scheduling problem is mapped to a Multiple-Choice Minimum-Cost Maximal Knapsack Packing problem, in which each device is represented as a class of possible workload choices, and the objective is to fill the required workload with minimum total cost. The paper provides a pseudo-polynomial dynamic-programming solution for arbitrary cost functions, as well as faster optimal algorithms for monotonic cost behaviors. Thus, *(MC)²MKP* tackles client participation as an energy-aware workload-assignment problem that minimizes total training cost while respecting device-specific workload limits.

MEC and *ECMTC* (28), proposed in this thesis, address client selection by jointly optimizing FL round time and energy consumption through workload assignment. Both methods decide not only which clients participate, but also how many training tasks

each client processes. MEC follows a lexicographic objective that first minimizes the round makespan and then minimizes total energy among schedules with that optimal makespan. In contrast, ECMTC reverses the priority by first minimizing total energy consumption and then minimizing makespan, while respecting a user-defined deadline. Both algorithms use dynamic programming to explore feasible task assignments from client-specific execution-time and energy profiles, and their optimality is proved. Thus, MEC and ECMTC tackle client selection as a time-energy-aware scheduling problem that assigns workloads to heterogeneous clients according to the desired optimization priority.

Overall, optimal scheduling-based methods provide strong guarantees for specific objectives such as makespan, energy consumption, or time-energy trade-offs. They are relevant because they explicitly model workload allocation rather than only deciding whether each client is selected. However, their objective scope is usually restricted to system-level criteria and does not include FL-specific aspects such as participation diversity, privacy-preserving class-distribution quality, historical learning behavior, or reliability-aware workload restriction. MetaCS-FL extends this line of work by using the optimal schedule generated by ECMTC as a structured initial solution and refining it with the LNS metaheuristic to optimize a broader objective that combines system efficiency with fairness, data awareness, reliability, and loss-based utility.

3.6 Metaheuristic-Based Methods

Metaheuristic methods are useful for complex client selection problems because they can explore large combinatorial search spaces and balance multiple objectives without requiring strong structural assumptions about the objective function. In FL, these methods have been used to select client subsets or deploy clients under multiple criteria, such as accuracy, energy, delay, reliability, fairness, mobility, and resource availability. Their main advantage is flexibility, especially when exact optimization becomes impractical or when several competing objectives must be considered simultaneously.

GWO-FL (100) addresses client selection in over-the-air FL under wireless and resource constraints. The method formulates selection as a multi-attribute optimization problem that considers model loss, energy consumption, computation and transmission delay, reliability, and fairness. Each candidate solution represents a set of clients, and the Grey Wolf Optimizer (GWO) searches for the best client subset by updating candidate solutions according to the alpha, beta, and delta wolves, which represent the best

solutions found so far. The selected clients then train locally and transmit their updates through over-the-air aggregation. Thus, GWO-FL tackles client selection as a metaheuristic multi-criteria optimization problem that balances learning quality, energy efficiency, latency, reliability, and fair participation in wireless FL.

On-Demand-FL (101) addresses client participation in highly dynamic FL environments where suitable devices may not be available in advance. Instead of only selecting from a fixed client pool, the method uses Docker and Kubernetes/Kubeadm to deploy FL clients on volunteer IoT and mobile devices whenever and wherever additional data or resources are needed. The deployment decision is formulated as a multiobjective optimization problem that considers resource capacity, battery, availability time, mobility, area location, data volume, learning quality, data diversity, and on-demand requests from orchestrators. A Genetic Algorithm (GA) is then used to approximate Pareto-efficient deployments while repairing infeasible solutions that violate resource or availability constraints. Thus, On-Demand-FL tackles client selection as an on-demand deployment and multicriteria selection problem that increases client availability, data volume, and data heterogeneity in dynamic FL settings.

Overall, metaheuristic-based methods are well suited for multi-objective FL selection problems, especially when the search space is large and exact optimization is impractical. However, existing approaches commonly search over client subsets or deployment decisions, rather than over client-task assignments. As a result, they usually decide which clients participate but not how much workload each selected client should process. MetaCS-FL differs by combining structured scheduling with Large Neighborhood Search (LNS): it starts from an ECMTC-generated workload assignment and then refines task allocations directly, allowing the server to jointly decide client participation and workload size while accounting for time, energy, diversity, class-distribution quality, reliability, and historical loss-based utility.

3.7 Overview and Positioning

Table 1 summarizes the capabilities of the reviewed approaches and highlights the position of MetaCS-FL with respect to the literature. The comparison focuses on whether each method explicitly addresses utility, time, energy, fairness or diversity, data awareness, workload control, and reliability or adaptivity. A checkmark indicates that the corresponding capability is part of the method's selection, aggregation, deployment, or

scheduling logic, rather than only an indirect consequence of the experimental setting.

Table 1: Summary of literature methods and supported capabilities. A checkmark indicates that the capability is explicitly addressed by the method.

Method	Utility Aware	Time Aware	Energy Aware	Fairness / Diversity	Data Aware	Workload Control	Reliability / Adaptivity
FedAvg (1)	–	–	–	–	–	–	–
FedSampling (78)	–	–	–	–	✓	✓	–
Favor (79)	✓	–	–	–	✓	–	✓
FedPNS (9)	✓	–	–	–	–	–	✓
AUCTION (80)	✓	–	–	–	✓	–	–
AdaFL (81)	✓	–	–	–	–	–	✓
Oort (46)	✓	✓	–	–	–	–	✓
CriticalFL (82)	✓	–	–	–	–	–	✓
FedCS (7)	–	✓	–	–	–	–	–
Ouyang and Liu (83)	✓	✓	✓	–	–	✓	✓
FedMCCS (5)	–	✓	✓	–	✓	–	✓
GREED (6)	–	✓	✓	–	–	–	–
EACS-FL (84)	–	✓	✓	✓	–	–	✓
ESCS (85)	✓	✓	✓	–	–	–	–
FedAECS (12)	✓	✓	✓	–	✓	–	–
FEPCS (86)	✓	–	✓	–	✓	✓	✓
Fed-RAC (87)	–	✓	–	–	–	–	✓
ELASTIC (10)	–	✓	✓	–	–	–	✓
FedAEB (88)	✓	✓	✓	–	✓	–	✓
RaFed (89)	✓	✓	–	–	–	–	✓
FedSynthesis (52)	–	–	✓	–	–	–	✓
DCSE (90)	✓	–	✓	–	–	–	✓
DOCS (91)	✓	–	–	–	–	–	✓
FedJoule (92)	✓	✓	✓	✓	✓	–	✓
AEDFL (93)	✓	–	–	–	–	–	✓
FedCAB (17)	✓	–	–	✓	✓	–	✓
CA-Fed (18)	✓	–	–	–	–	–	✓
TwoStageEAFL (19)	✓	✓	✓	–	–	–	✓
FedAWE (20)	✓	–	–	–	–	–	✓
FLICS-OPT (21)	–	–	–	✓	✓	–	✓
RIFLES (22)	–	✓	–	✓	–	–	✓
q-FedAvg (95)	✓	–	–	✓	–	–	–
ECSM (96)	✓	–	–	✓	–	–	✓
DivFL (15)	✓	–	–	✓	–	–	–
FedTune-SGM (97)	✓	–	–	✓	–	–	–
xCS (98)	✓	–	–	✓	–	–	✓
Fed-LBAP / MinCost (99)	✓	✓	–	–	✓	✓	–
OLAR (8)	–	✓	–	–	–	✓	–
$(MC)^2MKP$ (60)	–	–	✓	–	–	✓	–
MEC / ECMTC (28)	–	✓	✓	–	–	✓	–
GWO-FL (100)	✓	✓	✓	✓	–	–	✓
On-Demand-FL (101)	✓	–	✓	✓	✓	–	✓
MetaCS-FL (29)	✓	✓	✓	✓	✓	✓	✓

Under the capability definitions adopted in Table 1, MetaCS-FL is the only method with a checkmark in all categories. This reflects its breadth: it combines learning-oriented utility, execution time, energy consumption, participation diversity, data-distribution awareness, explicit workload control, and reliability-aware adaptation within a client-task scheduling framework. This broader coverage does not imply that MetaCS-FL dominates every specialized method in its individual objective, theoretical guarantee, computational cost, or deployment setting. Instead, it identifies the integration of these separate dimensions as the main distinction of the proposed framework.

The literature shows that client selection has been studied from complementary per-

spectives, but most approaches cover only part of the design space addressed in this thesis. FedAvg provides the standard random-selection baseline, but it does not explicitly address client heterogeneity, resource constraints, fairness, adaptivity, data distribution, or workload control. Statistical and utility-aware methods improve convergence under non-IID data by prioritizing representative data, useful gradients, high-loss clients, or contribution estimates. However, they usually treat participation as a binary decision and rarely determine how much workload each selected client should process.

Resource-aware methods reduce latency, energy consumption, communication costs, or carbon emissions, and some of them consider energy budgets, wireless conditions, batch-size adaptation, power modes, or historical resource estimates. Nevertheless, many of these methods remain focused on a limited system-level trade-off, such as time-energy, energy-accuracy, or carbon-aware participation. Only a smaller group explicitly controls the workload assigned to each client, and even these methods usually do not combine workload allocation with fairness, data-distribution quality, reliability-aware capacities, and historical loss-based utility.

Dynamic-availability and asynchronous methods address client intermittency, late participation, non-stationary availability, availability forecasting, or decentralized coordination. These works are relevant because cross-device FL systems rarely operate with a fixed and permanently available client pool. However, they typically adapt sampling, aggregation, eligibility, or backup-client decisions, rather than solving a server-side client-task scheduling problem. In contrast, MetaCS-FL explicitly updates the candidate set, incorporates late-joining clients after profiling, reuses stored profiles for returning clients, and adjusts effective task capacities according to availability and completion history.

Fairness, diversity, explainability, and trustworthy-selection methods promote broader participation, reduce performance disparities, improve update representativeness, and justify why clients are selected. However, these approaches are often implemented through aggregation weights, contribution scores, update-diversity objectives, incentive mechanisms, or ranking policies. They generally do not optimize fairness together with execution time, energy consumption, data-distribution quality, reliability, and workload feasibility. MetaCS-FL instead embeds fairness directly into the scheduling objective through a recent-participation diversity score, allowing diversity to be balanced against system and learning objectives.

Optimal scheduling methods are particularly relevant because they move beyond binary client selection and explicitly assign data volumes or task counts to heterogeneous

clients. Fed-LBAP, OLAR, $(MC)^2MKP$, MEC, and ECMTC provide strong guarantees for specific workload-assignment objectives, such as makespan minimization, energy minimization, or time-energy lexicographic optimization. However, their objective scope is restricted mainly to system-level criteria. MetaCS-FL builds on this line of work by using ECMTC as a structured initial solution and then refining it through LNS to account for FL-specific criteria that optimal schedulers do not jointly address, including diversity, class-distribution quality, reliability-aware capacities, and historical loss-based utility.

Metaheuristic-based methods provide flexibility for multi-objective selection problems and can handle large combinatorial spaces. GWO-FL uses a Grey Wolf Optimizer to search for client subsets under learning, energy, delay, reliability, and fairness criteria, while On-Demand-FL uses a Genetic Algorithm to deploy clients under resource, mobility, availability, data-volume, and diversity objectives. However, these methods commonly operate at the level of client subsets or client deployment decisions. MetaCS-FL differs by applying metaheuristic search directly to client-task assignments: the server decides not only which clients participate but also how much training or testing workload each selected client processes.

Despite these advances in the literature, several limitations remain. First, many approaches rely on binary client selection, where selected clients process a fixed, full, or externally defined workload. Second, most methods optimize a narrow set of objectives, such as accuracy, latency, energy, carbon emissions, availability, or trustworthiness, without jointly considering system efficiency, fairness, data balance, reliability, and learning behavior. Third, historical client behavior is often used for selection probabilities, availability estimation, or contribution ranking, but is not always integrated into workload feasibility and multi-objective scheduling. Fourth, data-distribution awareness is frequently absent, indirect, or not combined with privacy-preserving disclosure. Finally, loss information is commonly used as an isolated utility or fairness signal rather than as one component of a broader scheduling formulation.

To address these limitations, we propose *MetaCS-FL* (29), a flexible and event-driven metaheuristic-based framework for synchronous cross-device FL. MetaCS-FL formulates client selection as a client-task scheduling problem, enabling fine-grained workload control by assigning different amounts of training or testing effort to selected clients. Its unified objective jointly considers makespan, energy consumption, client diversity, class-distribution quality, and historical loss-based utility. Reliability-aware task capacities reduce the risk of assigning excessive workloads to clients with unstable availability or poor

completion history, while privacy-preserving class histograms allow the server to account for data-distribution quality without requiring exact local class distributions. By combining ECMTC-based initialization, LNS-based refinement, event-driven triggers, selection reuse, asynchronous profiling for late joiners, and reliability-aware workload restriction, MetaCS-FL bridges optimal workload scheduling and metaheuristic multi-objective client selection in a single framework.

3.8 Design Scope

The methods developed in this thesis target centralized, synchronous cross-device FL deployments in which a server manages diverse, resource-constrained clients, including mobile and edge devices. Raw data remain local, while the server handles model distribution, client selection, workload assignment, and aggregation. This setting is relevant when differences in computation and communication speed, energy consumption, availability, and data distributions make uniform participation or fixed workloads inefficient.

A centralized approach is considered because the model-centric FL workflow assigns global coordination to the server. Extending this coordinator with scheduling decisions allows it to distribute a user-defined workload based on client-specific time, energy, reliability, participation, and data-distribution information. This choice does not establish that centralized FL is intrinsically preferable to decentralized or hierarchical alternatives, but defines the system boundary of the proposed scheduling problems. Centralization may create a coordination bottleneck and concentrate overhead at the server, so scalability and server-side costs are evaluated in Sections 6.2.4 and 6.2.5.

Synchronous execution is adopted as communication rounds provide a well-defined decision and observation interval. Selected clients receive the same global model version, their performance can be compared within the same interval, and the server can enforce a deadline before aggregation. These properties support the round-level client-task scheduling formulation in Section 2.3.2. As discussed in Section 2.2.3, asynchronous execution can reduce blocking but introduces stale updates and shifts the problem toward admission control, concurrency regulation, buffering, and update filtering. Adapting the proposed algorithms to asynchronous, decentralized, hierarchical, or hard-real-time environments would require a different formulation and lies outside the present scope.

The task-oriented formulation is a computational abstraction for controlling workload granularity. A task represents a configured unit of local data processing, has no precedence

constraints, and is atomic below the chosen scheduling granularity. Equal-sized tasks are not assumed to contain statistically identical or IID data. MEC and ECMTC use this abstraction to optimize execution time and energy without interpreting the semantic contents of individual data slices. MetaCS-FL additionally maintains per-class task counts from exact or DP-perturbed class histograms and uses them to evaluate class coverage and balance in a workload allocation. Thus, common scheduling granularity does not remove or contradict the statistical heterogeneity of client data.

MetaCS-FL combines five normalized objectives, *i.e.*, makespan, energy consumption, recent-participation diversity, class-distribution quality, and historical loss-based utility, through scalarization, while reliability modifies feasible client task capacities. A scalarized score gives the metaheuristic a criterion for comparing schedules while making the trade-offs explicit through configurable weights. The fixed experimental weights provide a controlled basis for comparison and emphasize criteria absent from the time- and energy-aware ECMTC initial solution. In deployment, weights may be configured by application priorities, while adaptive weighting and objective-ablation analyses remain extensions.

4 MEC and ECMTC: Optimal Time and Energy-Aware Client Selection Algorithms

This chapter focuses on two system-level dimensions of FL client selection: execution time and energy consumption. The execution time of a synchronous training round is limited by the slowest selected client, commonly referred to as the straggler (8, 102, 99), in the FL-specific sense defined in Section 2.2.3. At the same time, energy consumption has become a relevant concern in FL due to battery limitations in edge devices (103), energy availability constraints (104), and environmental impact (105). Although these dimensions directly affect the feasibility and efficiency of FL systems, few works jointly optimize time and energy while providing optimality guarantees.

To address this gap, this chapter presents two client selection algorithms, *MEC* and *ECMTC*, that jointly optimize training time and energy consumption in cross-device FL systems. Instead of deciding only which clients participate in a round, the proposed algorithms also determine how much data each selected client should use locally. MEC prioritizes the minimization of training time and, subsequently, energy consumption, whereas ECMTC reverses this priority while satisfying a deadline constraint. Section 4.1 models client selection in cross-device FL systems as a task scheduling problem. Section 4.2 introduces the proposed algorithms and proves their optimality. Section 4.3 presents a preliminary experimental analysis comparing them with strategies from the literature.

4.1 Client Selection as a Task Scheduling Problem

This section formalizes FL client selection as a task scheduling problem. Clients are modeled as heterogeneous resources, while the workload of a training round is represented as a set of tasks. This abstraction supports the definition of optimization objectives based on execution time, energy consumption, and deadline constraints.

4.1.1 Problem Definition

Consider n heterogeneous resources organized in a set $\mathcal{R} = \{1,2,\ldots,n\}$. These resources are required to train machine learning models with a workload size of $T \in \mathbb{N}$. This workload contains equal-sized, independent, and atomic tasks, which represent slices of local data. Equal-sized means that every task contains the same number of samples from the scheduler's perspective and does not imply that the samples have identical classes or statistical distributions. Independence means that tasks have no precedence constraints, whereas atomicity means that a task cannot be divided below the selected scheduling granularity. Thus, the abstraction specifies workload quantity, while the statistical content remains local to each client and may be non-IID. MEC and ECMTC do not use this content in their system-level objectives, and the experiments in this chapter use balanced partitions, while Chapter 5 introduces class-distribution-aware selection. Table 2 summarizes the notation used throughout the definition of the optimization problems.

Table 2: Notation summary for the optimization problems.

Symbol	Meaning
n	Number of resources.
$\mathcal{R}$	Set of resources.
T	Size of the workload.
A_i	Set of feasible task capacities for resource i.
x_i	Number of tasks assigned to resource i.
P_i	Set of execution times of resource i.
E_i	Set of energy consumptions of resource i.
D	User-defined deadline.
X	Schedule that assign all tasks to resources.
X^*	Optimal schedule.
$C_{\max}$	Makespan (maximum execution time) of a schedule.
ΣE	Total energy consumption of a schedule.

Each resource $i \in \mathcal{R}$ can be assigned $A_i \subset \mathbb{N}$ tasks to compute. Two functions, $P_i : A_i \to \mathbb{R}_{\geq 0}$ and $E_i : A_i \to \mathbb{R}_{\geq 0}$, represent the execution time (or performance) and energy consumption, respectively, of resource i when computing its assigned number of tasks. For now, no assumptions are made regarding the behavior or shape of these functions. Let $\mathcal{A} = \{A_1,\ldots,A_n\}$, $\mathcal{P} = \{P_1,\ldots,P_n\}$, and $\mathcal{E} = \{E_1,\ldots,E_n\}$ denote the sets of possible assignments, time functions, and energy functions.

Let $X = \{x_1,\ldots,x_n\}$ be the schedule that assigns $x_i \in A_i$ tasks to each resource $i \in \mathcal{R}$. Based on these functions, we define two distinct optimization targets: the *makespan* $C_{\max}$

(Eq. (4.1)) and the *total energy consumption* ΣE (Eq. (4.2)).

$$C_{\max} \coloneqq \max_{i \in \mathcal{R}} P_i(x_i) \tag{4.1}$$

$$\Sigma E \coloneqq \sum_{i \in \mathcal{R}} E_i(x_i) \tag{4.2}$$

4.1.2 Illustrative Example

Consider an FL system composed of one server and three heterogeneous clients, as illustrated in Figure 1. The server must decide how to distribute six mini-batches of work (tasks) during a training round. Each client can be assigned between zero and six tasks, *i.e.*, $A_i = \{0, \ldots, 6\}$, $\forall i \in \mathcal{R}$.

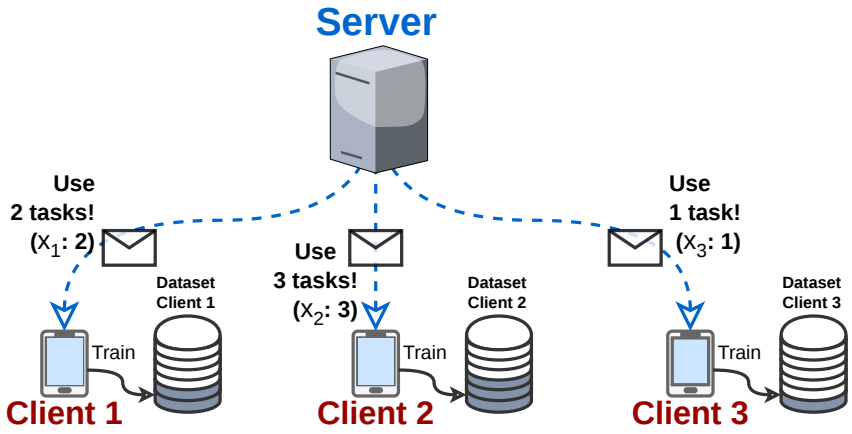

Figure 1: Interaction between the server and the selected clients concerning the distribution of tasks. Tasks represent slices of local data to be used by clients.

When the server requests that client 1 use two mini-batches (tasks) for training, as illustrated in Figure 1, the client uses a subset of its local data, which is never shared, to form the mini-batches, performs local training, and sends its updated model to the server.

Suppose that the server can estimate the time (P_i) and energy (E_i) required by each client i to process each task assignment capacity in A_i, as illustrated in Figure 2.

Consider a balanced task distribution strategy, where the workload is evenly allocated among the clients. In this example, with six tasks and three clients, each client would be assigned two tasks, resulting in a training time ($C_{\max}$) of 10 seconds, given by the slowest client, and a total energy consumption (ΣE) of 13.32 joules, given by the sum of the energy consumed by all clients (Figure 3, left-hand side).

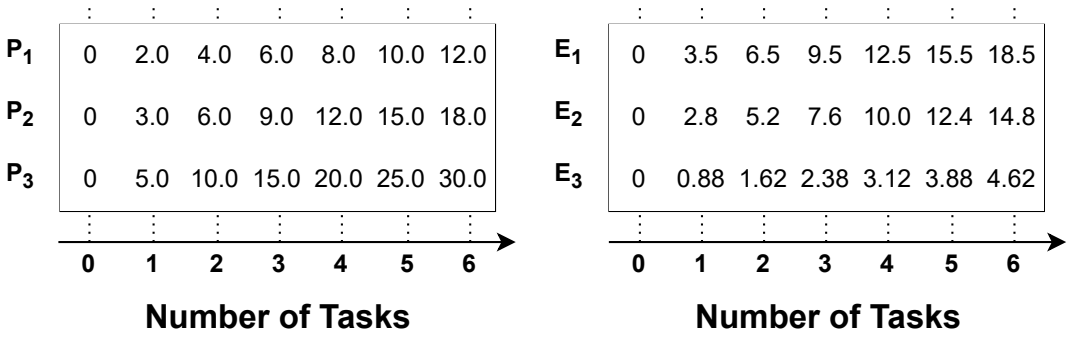

Figure 2: Performance and energy costs per client for each number of tasks.

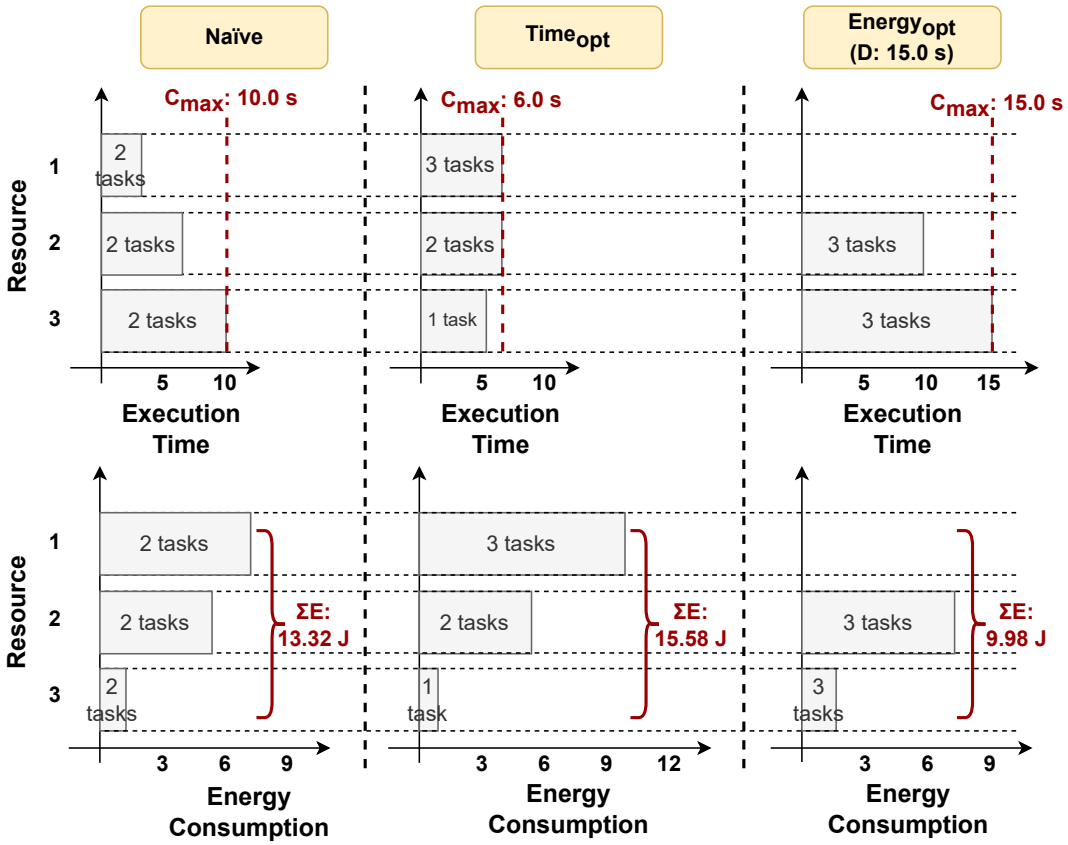

Figure 3: Examples of training times and energy consumption when scheduling six mini-batches (tasks) among the clients.

Now consider that the server aims to optimize training time first, followed by energy consumption. An optimal schedule would distribute three tasks for the first client, two for the second, and one for the third client, leading to $C_{\max} = 6$ seconds and $\Sigma E = 15.58$ joules (Figure 3, center).

Finally, consider that the server aims to optimize energy consumption first, followed by training time, subject to a deadline of $D = 15$ seconds. In this situation, no tasks would be given to the first client, while the other two would receive three tasks each,

leading to $C_{\max} = 15$ seconds and $\Sigma E = 9.98$ joules (Figure 3, right-hand side). If no deadline were imposed, a schedule with all tasks on the third client would lead to lower energy consumption ($\Sigma E = 4.62$ joules) but worse training time ($C_{\max} = 30$ seconds).

4.2 Optimal Time and Energy-Aware Scheduling Algorithms

The solutions for the multi-objective optimization problems presented in the illustrative example are based on a dynamic programming algorithmic structure (Algorithm 2).

Algorithm 2 Dynamic Programming Skeleton

Input: $\mathcal{R}$, T, $\mathcal{A}$, $\mathcal{P}$, and/or $\mathcal{E}$, D (default: $+\infty$).
Output: X^*, $C_{\max}$, and/or ΣE.

(I) Filtering
for $i = 1, \ldots, n$ **do** *//Initialize the minimal costs and partial solutions matrices*
  for $t = 0, \ldots, T$ **do**
    $I[i][t] \leftarrow \varnothing$
    (II) Initialization
for $j \in A_1$ **do**
  $I[1][j] \leftarrow j$
  (III) Find Solutions for the First Resource
for $i = 2, \ldots, n$ **do** *//Find the solutions for other resources*
  for $j \in A_i$ **do** *//Test all assignments to resource i*
    for $t = j, \ldots, T$ **do**
      (IV) Test New Solution
$t \leftarrow T$
for $i = n, \ldots, 1$ **do** *//Extract the optimal schedule X^**
  $j \leftarrow I[i][t]$; $x_i \leftarrow j$; $t \leftarrow t - j$
(V) Organize Final Solution

It computes partial solutions that optimize the problem with worst-case complexity of $\mathcal{O}(T^2 n)$ and includes five actions that can be executed differently depending on the problem being solved: (I) *Filtering*, (II) *Initialization*, (III) *Find Solutions for the First Resource*, (IV) *Test New Solution*, and (V) *Organize Final Solution*.

4.2.1 MEC

Given a problem instance ($\mathcal{R}$,T,$\mathcal{A}$,$\mathcal{P}$,$\mathcal{E}$), the goal of the *Minimal* ***M****akespan and* ***E****nergy* ***C****onsumption FL Schedule* (MEC) problem is to find an optimal schedule X^* that minimizes $C_{\max}$ and ΣE, in that order (Eq. (4.3)).

$$\textbf{lex min}_X \; C_{\max}, \Sigma E \tag{4.3a}$$

$$\textbf{subject to} \sum_{i \in \mathcal{R}} x_i = T, \tag{4.3b}$$

$$x_i \in A_i, \, \forall i \in \mathcal{R} \tag{4.3c}$$

With the lexicographic method (lex), a type of multi-objective optimization, the objective functions are arranged in order of importance. Once the order is set, each function is solved so that each subsequent problem does not degrade any of the previous solutions (106).

4.2.1.1 Solving MEC

A two-step approach is required to solve MEC (Algorithm 3). Firstly, the minimal makespan is computed (Algorithm 4) and then used as the deadline D when computing the minimal energy consumption (Algorithm 5). The second step provides the optimal schedule with minimal makespan and energy consumption, as proven in 4.2.1.2.

Algorithm 3 MEC

1: $\_, C_{\max}, \_ \leftarrow$ **MinMaxTime**$(\mathcal{R}, T, \mathcal{A}, \mathcal{P})$
2: $X^*, \_, \Sigma E \leftarrow$ **MinSumEnergy**$(\mathcal{R}, T, \mathcal{A}, \mathcal{P}, \mathcal{E}, C_{\max})$
3: **return** X^*, $C_{\max}$, ΣE

Step 1: Computing the Minimal Makespan. Consider $\mathcal{Z}_r^P(\tau)$ as an optimal solution value for a partial problem with the first r resources that schedule τ tasks. Assume that $\mathcal{Z}_r^P(\tau) \coloneqq \infty$ if no solution exists and that $\mathcal{Z}_0^P(0) \coloneqq 0$. Thus, $\mathcal{Z}_r^P(\tau)$ is defined in Eq. (4.4) and can be recursively computed following Eq. (4.5).

$$\mathcal{Z}_r^P(\tau) \coloneqq \min \left\{ \max_{i \in [1,r]} P_i(x_i) \,\middle|\, \sum_{i=1}^{r} x_i = \tau \right\} \tag{4.4}$$

$$\mathcal{Z}_r^P(\tau) = \min_{j \in A_r, j \leq \tau} \max \left(\mathcal{Z}_{r-1}^P(\tau - j), P_r(j) \right) \tag{4.5}$$

Algorithm 4 employs the properties of Eqs. (4.4) and (4.5). It computes all possible solutions for $\mathcal{Z}_1^P$ in action (III), then computes optimal solutions for an increasing number of resources in the main loop (Algorithm 2, line 9). The best solution for a given resource i and t tasks is defined by considering the previous best solution with $i-1$ resources

(based on Eq. (4.5)). The makespan of this partial solution is computed and stored in $P[i][t]$ by comparing the maximum between the previous best solution and the execution time for a given number of tasks in resource i. The minimal makespan is found by keeping the minimal solution in action (IV). The solution for $\mathcal{Z}_n^P(T)$ is returned in action (V).

Algorithm 4 MinMaxTime (Minimal Makespan)

1: (I) Filtering:
2: **for** $i = 1, \ldots, n$ **do** *//Allow only assignments that respect T*
3: $A_i \leftarrow \{j | j \in A_i \wedge j \leq T\}$
4: (II) Initialization: $P[i][t] \leftarrow \infty$
5: (III) Find Solutions for the First Resource: $P[1][j] \leftarrow P_1(j)$
6: (IV) Test New Solution:
7: $P_{\text{new}} \leftarrow \max(P[i-1][t-j], P_i(j))$
8: **if** $P_{\text{new}} < P[i][t]$ **then**
9: $P[i][t] \leftarrow P_{\text{new}}$; $I[i][t] \leftarrow j$ *//Best solution so far*
10: (V) Organize Final Solution: $C_{\max} \leftarrow P[n][T]$

Step 2: Computing the Minimal Energy Consumption. Consider $\mathcal{Z}_r^E(\tau, D)$ as an optimal solution value for a partial problem with the first r resources that schedule τ tasks while meeting the deadline D. Similar to the previous case, assume that $\mathcal{Z}_r^E(\tau,\cdot) \coloneqq \infty$ if no solution exists and that $\mathcal{Z}_0^E(0,\cdot) \coloneqq 0$. Thus, $\mathcal{Z}_r^E(\tau, D)$ is defined in Eq. (4.6) and can be recursively computed following Eq. (4.7).

$$\mathcal{Z}_r^E(\tau, D) \coloneqq \min \left\{ \sum_{i=1}^{r} E_i(x_i) \;\middle|\; \begin{array}{l} \sum_{i=1}^{r} x_i = \tau, \\ P_i(x_i) \leq D, i \in [1,r] \end{array} \right\} \tag{4.6}$$

$$\mathcal{Z}_r^E(\tau, D) = \min_{j \in A_r, j \leq \tau, P_r(j) \leq D} \mathcal{Z}_{r-1}^E(\tau - j, D) + E_r(j) \tag{4.7}$$

Algorithm 5, which is a deadline-constrained specialization of (MC)^{2}MKP (60), employs the properties of Eqs. (4.6) and (4.7) and discards in action (I) any task assignments that could surpass the deadline D. Moreover, only solutions with minimal energy consumption are kept in action (IV).

Each item class N_i in (MC)^{2}MKP corresponds to a client or resource $i \in \mathcal{R}$, and each item $j \in N_i$ corresponds to an admissible task assignment $j \in A_i$. The item weight w_{ij} becomes the number of assigned tasks, *i.e.*, $w_{ij} = j$, while the item cost c_{ij} becomes energy consumption $E_i(j)$. Consequently, cost matrix K of Algorithm 1 in (60) corresponds to energy matrix E in Algorithm 5, while both algorithms use matrix I to recover the assignment for each resource. Actions (II)–(V) retain the same dynamic programming

structure: initialization of partial solutions, construction of the base case for the first resource, evaluation of the additive recurrence, and backtracking to recover the assignment.

The difference is that Algorithm 5 adds the deadline filter in action (I), removing assignments j for which $P_i(j) > D$. Besides this filtering and the interpretation of the accumulated cost as energy consumption, it retains the exact-workload dynamic programming structure of Algorithm 1 of $(MC)^2MKP$ when applied to a Minimal Cost FL Schedule instance. Thus, Algorithm 5 inherits its additive recurrence and solution-recovery procedure while restricting the admissible assignments to those satisfying the deadline.

Algorithm 5 MinSumEnergy (Minimal Energy Consumption)

(I) Filtering:
for $i = 1,\ldots,n$ **do** *//Allow only assignments that respect T and D*
    $A_i \leftarrow \{j | j \in A_i \wedge j \leq T \wedge P_i(j) \leq D\}$
(II) Initialization: $E[i][t] \leftarrow \infty$
(III) Find Solutions for the First Resource: $E[1][j] \leftarrow E_1(j)$
(IV) Test New Solution:
$E_{\text{new}} \leftarrow E[i-1][t-j] + E_i(j)$
if $E_{\text{new}} < E[i][t]$ **then**
    $E[i][t] \leftarrow E_{\text{new}}$; $I[i][t] \leftarrow j$ *//Best solution so far*
(V) Organize Final Solution: $\Sigma E \leftarrow E[n][T]$

4.2.1.2 Proof of optimality

Theorem 1. *$\mathcal{Z}_n^P(T)$ yields the optimal solution to the Minimal Makespan Problem.*

Proof. We prove the stronger statement that, for every $r \in \{1,\ldots,n\}$ and every $\tau \in [0,T]$, $\mathcal{Z}_r^P(\tau)$ is the minimum makespan among all schedules that assign exactly τ tasks to the first r resources.

Base case. For $r = 1$, a feasible schedule assigning τ tasks to the first resource exists only when $\tau \in A_1$. Its makespan is necessarily $P_1(\tau)$, which is the value stored by action (III). If $\tau \notin A_1$, the state remains infinite. Hence, every state $\mathcal{Z}_1^P(\tau)$ is optimal.

Induction hypothesis. Assume that, for some $i \in \{1,\ldots,n-1\}$ and every $t \in [0,T]$, $\mathcal{Z}_i^P(t)$ is the minimum makespan for assigning exactly t tasks to the first i resources.

Induction step. Consider any schedule assigning τ tasks to the first $i+1$ resources. If the last resource receives $j \in A_{i+1}$ tasks, the first i resources must receive $\tau - j$ tasks. By the induction hypothesis, their smallest possible makespan is $\mathcal{Z}_i^P(\tau - j)$. Therefore, the smallest makespan among schedules using this value of j is $\max\big(\mathcal{Z}_i^P(\tau - j), P_{i+1}(j)\big)$.

Enumerating every admissible $j \leq \tau$ and taking the minimum gives exactly Eq. (4.5). Thus, $\mathcal{Z}_{i+1}^{P}(\tau)$ is optimal for every τ.

Conclusion. By induction, the statement holds for all $r \leq n$. In particular, $\mathcal{Z}_{n}^{P}(T)$ is the minimum-makespan schedule for all T tasks. □

The optimality of Algorithm 5 follows from the same optimal-substructure argument used for Algorithm 1 of (MC)^{2}MKP (60, Theorem 1). After action (I), every assignment satisfies $P_i(j) \leq D$, and the filter removes no assignment that could belong to a deadline-feasible schedule. For each pair (i,t), $E[i][t]$ stores the minimum accumulated energy for assigning exactly t tasks to the first i resources. Action (IV) enumerates every admissible $j \in A_i$ and retains the minimum-energy combination of $E[i-1][t-j]$ and $E_i(j)$. Hence, $E[n][T]$ is the minimum-energy schedule that assigns exactly T tasks within the deadline.

Theorem 2. *The schedule computed by MEC is optimal.*

Proof. *MinMaxTime* (Algorithm 4) finds a schedule with minimal makespan (Theorem 1), which is used to discard any task assignments that could surpass its value at action (I) of *MinSumEnergy* (Algorithm 5). Consequently, no schedule computed by MinSumEnergy can have a higher (due to filtering) or lower (due to the optimality of MinMaxTime) makespan. By the dynamic programming argument above, which specializes (MC)^{2}MKP (60, Theorem 1), MinSumEnergy finds the minimum-energy schedule among those with this makespan. Thus, the schedule computed by MEC (Algorithm 3) achieves minimal makespan and energy consumption and, therefore, it is optimal. □

4.2.2 ECMTC

Given a problem instance $(\mathcal{R},T,\mathcal{A},\mathcal{P},\mathcal{E},D)$, with a deadline $D \in \mathbb{R}_{\geq 0}$, the goal of the *Minimal **E**nergy **C**onsumption and **M**akespan FL Schedule under **T**ime **C**onstraint* (ECMTC) problem is to find an optimal schedule X^* that minimizes ΣE and $C_{\max}$, in that order, while meeting the deadline (Eq. (4.8)).

$$\textbf{lex min}_X \ \Sigma E, C_{\max} \tag{4.8a}$$

$$\textbf{subject to} \sum_{i \in \mathcal{R}} x_i = T, \tag{4.8b}$$

$$C_{\max} \leq D, \tag{4.8c}$$

$$x_i \in A_i, \forall i \in \mathcal{R} \tag{4.8d}$$

Similarly to MEC, ECMTC uses the lexicographic approach to prioritize objectives.

4.2.2.1 Solving ECMTC

Unlike MEC, ECMTC can be computed in a single pass. Consider $\mathcal{Z}_r^{EP}(\tau, D)$ as an optimal solution pair (ΣE,$C_{\max}$) for a partial problem with the first r resources that schedules τ tasks while meeting the deadline D. Assume that $\mathcal{Z}_r^{EP}(\tau,\cdot) \coloneqq (\infty,\infty)$ if no solution exists and that $\mathcal{Z}_0^{EP}(0,\cdot) \coloneqq (0,0)$. Thus, $\mathcal{Z}_r^{EP}(\tau, D)$ is defined in Eq. (4.9) and can be recursively computed following Eq. (4.10).

$$\begin{aligned} \mathcal{Z}_r^{EP}(\tau, D) \coloneqq \ \text{lex min} \Bigg\{ & \sum_{i=1}^{r} E_i(x_i), \max_{i \in [1,r]} P_i(x_i) \\ & \text{such that} \sum_{i=1}^{r} x_i = \tau \wedge P_i(x_i) \leq D, i \in [1,r] \Bigg\} \end{aligned} \tag{4.9}$$

$$\begin{aligned} \mathcal{Z}_r^{EP}(\tau, D) = \ & \text{lex} \min_{j \in A_r, j \leq \tau, P_r(j) \leq D} e + E_r(j), \max(p, P_r(j)) \\ & \text{where } (e,p) = \mathcal{Z}_{r-1}^{EP}(\tau - j, D) \end{aligned} \tag{4.10}$$

Algorithm 6 employs the properties of Eqs. (4.9) and (4.10). It does the same filtering as MinSumEnergy, and it combines actions (II), (III), and (V) of MinMaxTime and MinSumEnergy. Its main difference lies in action (IV). The best solution for a given resource i and t tasks is still defined based on the previous best solution with $i-1$ resources. However, two conditions can lead to a better solution: (i) the energy consumption of the new solution is smaller than that of the current solution, or (ii) they both have the same energy consumption, and the new solution has a shorter execution time. By comparing the energy consumption and execution time of the previous best and current solutions and keeping their lexicographic minimum, $\mathcal{Z}_i^{EP}(t,D)$ is computed, and the minimum energy consumption and makespan pair is found. The optimality of the schedule provided by ECMTC, with minimal energy consumption and makespan, is proven in 4.2.2.2.

Algorithm 6 ECMTC

(I) Filtering:
for $i = 1,\ldots,n$ **do** *//Allow only assignments that respect T and D*
    $A_i \leftarrow \{j | j \in A_i \wedge j \leq T \wedge P_i(j) \leq D\}$
(II) Initialization: $E[i][t] \leftarrow \infty$; $P[i][t] \leftarrow \infty$
(III) Find Solutions for the First Resource: $E[1][j] \leftarrow E_1(j)$; $P[1][j] \leftarrow P_1(j)$
(IV) Test New Solution:
$E_{\text{new}} \leftarrow E[i-1][t-j] + E_i(j)$
$P_{\text{new}} \leftarrow \max(P[i-1][t-j], P_i(j))$
if $(E_{\text{new}} < E[i][t])$ **or** $(E_{\text{new}} = E[i][t]$ **and** $P_{\text{new}} < P[i][t])$ **then**
    $E[i][t] \leftarrow E_{\text{new}}$; $P[i][t] \leftarrow P_{\text{new}}$; $I[i][t] \leftarrow j$ *//Best solution so far*
(V) Organize Final Solution: $\Sigma E \leftarrow E[n][T]$; $C_{\max} \leftarrow P[n][T]$

4.2.2.2 Proof of optimality

Theorem 3. *$\mathcal{Z}_n^{EP}(T)$ yields an optimal solution to the ECMTC problem.*

Proof. We prove the stronger statement that, for every $r \in \{1,\ldots,n\}$ and every $\tau \in [0,T]$, $\mathcal{Z}_r^{EP}(\tau,D)$ is lexicographically minimal among all deadline-feasible schedules that assign exactly τ tasks to the first r resources.

Base case. For $r = 1$, a deadline-feasible schedule assigning τ tasks exists only when $\tau \in A_1$ and $P_1(\tau) \leq D$. Its unique objective pair is $\big(E_1(\tau),P_1(\tau)\big)$, which action (III) stores. Otherwise, the state remains (∞,∞). Hence, every state $\mathcal{Z}_1^{EP}(\tau,D)$ is optimal.

Induction hypothesis. Assume that, for some $i \in \{1,\ldots,n-1\}$ and every $t \in [0,T]$, $\mathcal{Z}_i^{EP}(t,D)$ is the lexicographically minimal energy–makespan pair for assigning exactly t tasks to the first i resources while meeting D.

Induction step. Consider any deadline-feasible schedule assigning τ tasks to the first $i+1$ resources. If resource $i+1$ receives $j \in A_{i+1}$ tasks, with $j \leq \tau$ and $P_{i+1}(j) \leq D$, the first i resources receive $\tau-j$ tasks. Let $(e,p) = \mathcal{Z}_i^{EP}(\tau-j,D)$. By the induction hypothesis, this prefix pair is lexicographically minimal. For the fixed assignment j, extending it produces $\big(e + E_{i+1}(j), \max\{p,P_{i+1}(j)\}\big)$. This extension preserves lexicographic order: lower prefix energy yields lower total energy, and, when energies are equal, a no-larger prefix makespan yields a no-larger resulting makespan. Taking the lexicographic minimum over admissible j therefore yields exactly Eq. (4.10), so $\mathcal{Z}_{i+1}^{EP}(\tau,D)$ is optimal for every τ.

Conclusion. By induction, the statement holds for all $r \leq n$. In particular, $\mathcal{Z}_n^{EP}(T,D)$ is the lexicographically minimum energy–makespan pair among all deadline-feasible schedules and therefore solves ECMTC optimally. □

4.2.3 Pareto Relationship between MEC and ECMTC

Figure 4 illustrates the interactions between the different problems by comparing the makespan (horizontal axis) and total energy consumption (vertical axis) of varied solutions.

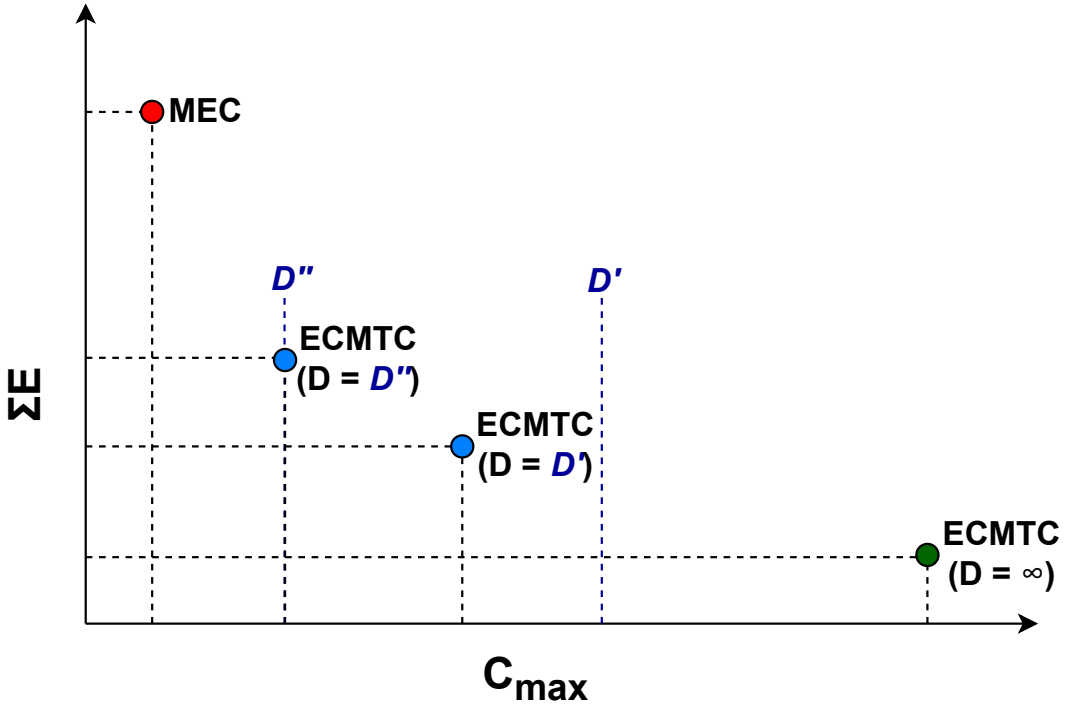

Figure 4: Relation between MEC and ECMTC for a given problem instance.

The MEC solution (red solid point) achieves the minimum makespan and, among schedules with that makespan, the lowest energy consumption. The ECMTC solution with $D \approx \infty$ (green solid point) achieves the minimum energy consumption and, among schedules with that energy consumption, the lowest makespan. Between these two extremes, ECMTC solutions with specific deadlines (blue solid points) achieve the minimum energy consumption within each deadline and, among schedules with that energy consumption, the lowest makespan. All these points lie on the Pareto frontier, meaning that neither metric can be improved without worsening the other.

4.3 Preliminary Experimental Analysis

This section presents a preliminary evaluation of MEC and ECMTC against representative client selection strategies: ELASTIC (10), FedAECS (12), OLAR (8), and $(MC)^2$MKP (60). The goal is to assess how the proposed scheduling-based algorithms perform under con-

trolled simulation scenarios and in a real-world FL deployment using Flower (53) and Grid'5000 (107). Further details on the data and code used to run and analyze the experiments are available in the reproducibility instructions.

4.3.1 Experimental Setup

4.3.1.1 Benchmark algorithms

The evaluation compares the proposed algorithms, MEC and ECMTC, with four client selection strategies from the literature: ELASTIC (10), FedAECS (12), OLAR (8), and $(MC)^2MKP$ (60). In the real-world experiments, FedAvg (1), implemented through random client selection, is also included as a baseline.

4.3.1.2 Simulation-based configuration

The simulation-based experiments emulate resource heterogeneity by generating execution time, energy consumption, and accuracy values for each task assignment and resource. Three cost behaviors are considered: *linear*, $n \cdot \log(n)$, and *random*, as summarized in Table 3. The linear and $n \cdot \log(n)$ configurations follow the same monotonic structure and coefficient-generation procedure, and therefore tend to produce similar qualitative trends. The $n \cdot \log(n)$ configuration is retained because it introduces increasing marginal costs as more tasks are assigned to the same resource, making workload-concentration effects more visible.

Incremental random seeds are used to ensure reproducibility. The generated accuracy values are min-max normalized to the [0,1] range before being used by the algorithms.

Table 3: Functions used to simulate the resource heterogeneity.

Function	Behavior
linear	$f(x) = \alpha + \beta \cdot x$, where $x = t, \forall t \in [0..T]$ $\alpha,\beta \in [1, 10), \forall i \in \mathcal{R}$ (time) $\alpha,\beta \in [0.32, 3.2), \forall i \in \mathcal{R}$ (energy) $\alpha,\beta \in [0.2, 2), \forall i \in \mathcal{R}$ (accuracy)
$n \cdot \log(n)$	$f(x) = \alpha + \beta \cdot x \cdot \log(x+1)$, where $x = t, \forall t \in [0..T]$ $\alpha,\beta \in [1, 10), \forall i \in \mathcal{R}$ (time) $\alpha,\beta \in [0.32, 3.2), \forall i \in \mathcal{R}$ (energy) $\alpha,\beta \in [0.2, 2), \forall i \in \mathcal{R}$ (accuracy)
random	$f(x) = x$, where $x \in [0..T)$

4.3.1.3 Simulation-based experiment sets

Three sets of simulation-based experiments are considered.

Experiment Set ES_1 evaluates the makespan, energy consumption, and accuracy of the schedules produced by each algorithm. It considers 10 and 100 resources, workloads ranging from 1,000 to 5,000 tasks in steps of 100, and the three heterogeneity functions.

Experiment Set ES_2 evaluates the elapsed time required by each algorithm to compute a schedule. It is divided into two groups. In G_1, the number of resources is fixed at 100, while the number of tasks varies from 200 to 2,000 in steps of 200. In G_2, the number of tasks is fixed at 2,000, while the number of resources varies from 10 to 100 in steps of 15. Both groups use the *linear* function, and each measurement is based on 20 samples of five executions per algorithm.

Experiment Set ES_3 evaluates the trade-off between makespan and energy consumption for ECMTC. It considers MEC and ECMTC, 10 and 100 resources, workloads ranging from 1,000 to 5,000 tasks in steps of 100, and the three heterogeneity functions. The reference deadline is defined as the optimal makespan obtained by MEC, while relaxed deadlines range from 25% to 200% above this reference value, in steps of 25%.

4.3.1.4 Real-world configuration

The real-world experiments are implemented using Flower (53). The FL system consists of one server and 50 clients, each running on a dedicated Grid'5000 node (107). Accordingly, the evaluation isolates each client-side FL workload from co-located user applications, and resource sharing and external computational contention are not emulated. Each node is equipped with an Intel Xeon Gold 5220 CPU with 18 cores, 96 GiB of RAM, and no accelerators. To emulate client heterogeneity, the number of CPU cores available to each client is restricted. The 50 clients are divided into ten groups of five clients, where each group uses i cores, with $i \in \{1,2,4,6,8,10,12,14,16,18\}$.

The clients train MobileNetV2 (108), a lightweight convolutional neural network, on CIFAR-10 (25), which contains 32×32 color images organized into ten classes. The dataset is divided into balanced and disjoint local partitions using the dataset-splitter tool[1]. Each partition is stored locally and accessed only by its corresponding client, reflecting the data-locality assumption of FL. As a result, each client has 1,000 images for training and 200 images for testing, totaling 50,000 training images and 10,000 testing

[1]dataset-splitter – https://github.com/alan-lira/dataset-splitter

images. For each client, the task-assignment capacity ranges from zero to the number of available local training images, in increments of 100.

4.3.1.5 FL settings

The experiments run for 100 communication rounds. The batch size is set to 32, and the number of local epochs is set to 5. FedAvg (1), implemented as a weighted average, is used by the server to aggregate the model updates produced by the selected clients. PowerJoular (109) is used to measure energy consumption during training and testing.

During training, the evaluated client selection algorithms are Random (FedAvg), MEC, ECMTC, ECMTC_D15, OLAR, $(MC)^2$MKP, ELASTIC, and FedAECS. During testing, Random selection is used to evaluate the quality of the global model. Random selects 50% of the available clients for training and 20% for testing. For ECMTC and $(MC)^2$MKP, 80% of the available clients are randomly sampled before selection in each round to avoid repeatedly selecting the same energy-optimal subset. ECMTC uses no deadline, whereas ECMTC_D15 uses a 15-second training deadline.

4.3.1.6 Profiling and scheduling overhead

In the real-world experiments, the execution time, energy consumption, and accuracy associated with each task assignment are unknown before FL execution. Therefore, the first communication round profiles all 50 clients using an equal number of tasks. From the second round onward, the algorithms use historical measurements collected in previous rounds. After each successful training or testing execution, the client returns its newly observed metrics, and the server refreshes the corresponding profile before subsequent selection decisions. Profiling is therefore updated for participating clients throughout the FL execution rather than remaining fixed after the first round. Since only a subset of assignment capacities is observed during execution, linear interpolation or extrapolation estimates the missing performance values.

Client selection overhead is mitigated by overlapping selection with training. From round $r \geq 2$, the selection for round $r+1$ is performed concurrently with the execution of round r. Let ST denote the client selection time, TT the training makespan, CT the maximum communication time, and IT the idle time before the next round. The idle time is defined as $IT = ST-(TT+CT)$ if $ST > TT+CT$, and $IT = 0$ otherwise. Consequently, selection adds no delay to the next-round critical path when $ST \leq TT + CT$, and only a

positive IT is exposed as additional waiting time.

4.3.1.7 Workload settings

Two real-world workload settings are evaluated. The first schedules 12,500 training tasks and 2,500 testing tasks per round, while the second schedules 25,000 training tasks and 5,000 testing tasks per round. Each task corresponds to one local image. In each round, each selected client randomly samples a subset of its local data according to the number of tasks assigned to it.

4.3.1.8 Performance metrics

The evaluation considers makespan, total energy consumption, accuracy, client selection overhead, and client idle time. For the simulation-based experiments, the main metrics are makespan, energy consumption, accuracy, and algorithm execution time. For the real-world experiments, the main metrics are the mean number of selected clients, total training time, total energy consumption, final test accuracy, client selection time, communication time, and idle time.

4.3.2 Simulation-Based Results

4.3.2.1 Impact on makespan, energy, and accuracy

Figure 5 shows the makespan, energy consumption, and accuracy obtained for different numbers of tasks. Tables 4 and 5 report the results for the smallest and largest task configurations, corresponding to 1,000 and 5,000 tasks, respectively.

Table 4: ES_1 results for 100 $n \cdot \log(n)$ resources and 1,000 tasks.

Client Selection Algorithm	Makespan (s)	Energy Consumption (J)	Accuracy
MEC	105.83	3,144.66	0.0002
ECMTC	2,423.59	2,003.96	0.00003
OLAR	105.83	3,144.66	0.0002
$(MC)^2MKP$	2,423.59	2,003.96	0.00003
ELASTIC	244.34	4,173.18	0.0006
FedAECS	67,899.87	21,727.96	0.1502

For makespan, MEC and OLAR achieve the best results, with overlapping curves because both produce optimal schedules for this objective. MEC reduces the makespan

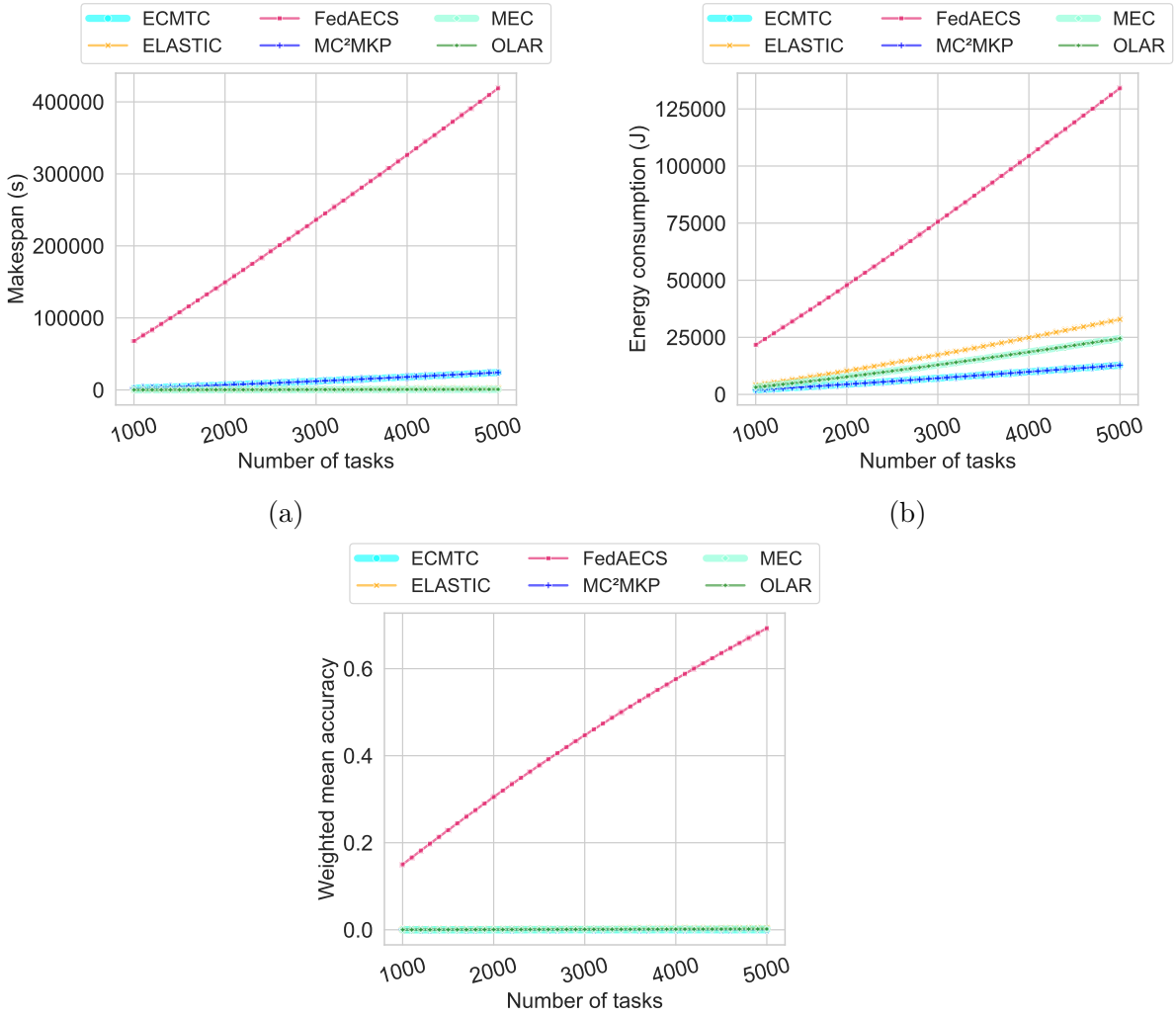

Figure 5: ES_1 results for 100 $n \cdot \log(n)$ resources regarding makespan (a), energy consumption (b), and accuracy (c) (1,000 to 5,000 tasks scheduled).

Table 5: ES_1 results for 100 $n \cdot \log(n)$ resources and 5,000 tasks.

Client Selection Algorithm	**Makespan (s)**	**Energy Consumption (J)**	**Accuracy**
MEC	777.98	24,526.49	0.0018
ECMTC	24,178.32	12,787.26	0.00003
OLAR	777.98	24,526.49	0.0018
$(MC)^2MKP$	24,178.32	12,787.26	0.00003
ELASTIC	1,941.03	32,878.40	0.0046
FedAECS	418,898.89	134,047.64	0.6930

by 40.08% compared with ELASTIC, which is the second-best result.

For energy consumption, ECMTC and $(MC)^2MKP$ achieve the best results, and their curves also overlap because both produce optimal schedules for this objective. For 5,000

tasks, ECMTC consumes 52.14% less energy than OLAR, the second-best literature algorithm in terms of energy consumption.

For accuracy, FedAECS outperforms the other algorithms across all task counts. However, this gain occurs because a single resource processes all tasks, which substantially increases both makespan and energy consumption.

4.3.2.2 Client selection overhead

Table 6 reports the minimum and maximum execution times required by each algorithm to schedule 2,000 tasks on 100 resources, corresponding to the worst-case configuration among the simulation-based experiments. These values are wall-clock seconds for one scheduling decision, not cumulative times across FL communication rounds.

Table 6: ES_2 minimum and maximum per-decision execution times in seconds for scheduling 2,000 tasks to 100 *linear* resources.

Client Selection Algorithm	**First Group (G_1)**		**Second Group (G_2)**	
	Min.	**Max.**	**Min.**	**Max.**
MEC	86.42	102.99	86.61	90.78
ECMTC	159.52	193.81	159.99	161.8
OLAR	0.012	0.015	0.012	0.013
$(MC)^2MKP$	52.13	62.86	52.03	53.17
ELASTIC	0.005	0.006	0.005	0.007
FedAECS	0.53	0.71	0.53	0.54

MEC and ECMTC require more time than the other algorithms. Compared with $(MC)^2MKP$, MEC and ECMTC take 40.13 seconds and 130.95 seconds longer, respectively, to compute a schedule. These differences correspond to increases of 63.84% and 208.31%. However, both algorithms optimize two objectives, and this overhead can be reduced in real-world scenarios where not all assignment capacities are known beforehand.

4.3.2.3 Time-energy trade-off

Table 7 summarizes the trade-off between makespan and energy consumption for 100 *linear* resources and 5,000 tasks. The percentage changes reported for ECMTC are computed relative to MEC. In this experiment, MEC provides the minimum-makespan reference solution, while ECMTC is evaluated under progressively relaxed deadlines. This setup allows us to analyze how much energy can be saved when the scheduler is allowed to increase training time.

Table 7: ES_3 results for 100 *linear* resources.

Client Selection Algorithm	Relaxed Deadline	Makespan (s)	Energy Consumption (J)
MEC	—	196.0	6,188.65
ECMTC	$125\% \rightarrow D = 245$	244.89 (+24.94%)	4,607.88 (−25.54%)
	$150\% \rightarrow D = 294$	294.0 (+50.0%)	3,878.75 (−37.32%)
	$175\% \rightarrow D = 343$	342.97 (+74.98%)	3,505.1 (−43.36%)
	$200\% \rightarrow D = 392$	391.94 (+99.97%)	3,238.61 (−47.67%)
	$225\% \rightarrow D = 441$	440.81 (+124.9%)	3,064.85 (−50.48%)
	$250\% \rightarrow D = 490$	489.89 (+149.94%)	2,936.56 (−52.55%)
	$275\% \rightarrow D = 539$	538.77 (+174.88%)	2,835.75 (−54.18%)
	$300\% \rightarrow D = 588$	587.87 (+199.93%)	2,756.81 (−55.45%)

Figure 6 shows the resulting time-energy trade-off curve. The curve exhibits a convex-like decreasing behavior: relaxing the deadline initially yields substantial energy reductions, but the marginal savings become progressively smaller as the allowed makespan increases. In continuous terms, this behavior can be interpreted through the curvature of the function relating energy consumption to makespan, where the second derivative captures how rapidly the slope changes. Since the deadlines are discrete, the knee point is identified by the largest change in slope between consecutive points, *i.e.*, the region where the discrete approximation of the second derivative is most pronounced.

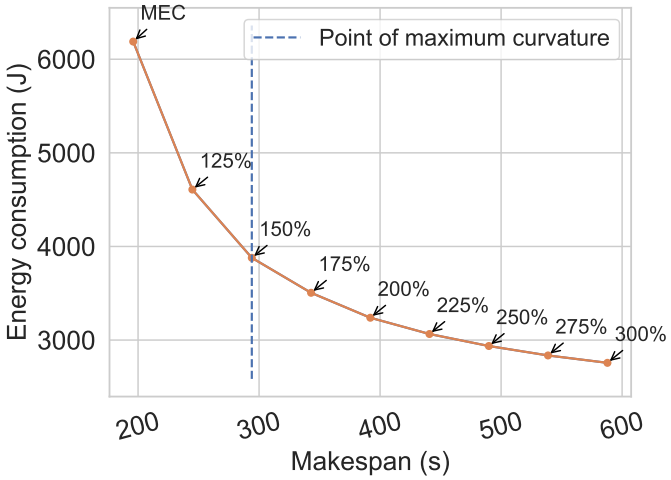

Figure 6: ES_3 trade-off curve for 100 *linear* resources.

The dashed vertical line marks the knee point. In the evaluated configuration, this point occurs when the deadline is set to 150% of MEC's optimal makespan. At this deadline, ECMTC reduces energy consumption by 37.32% compared with MEC, while increasing makespan by 50%. Beyond this point, further deadline relaxation still reduces energy consumption, but with smaller marginal gains. Therefore, the 150% deadline

provides the most balanced trade-off among the evaluated configurations.

4.3.3 Real-World Results

4.3.3.1 Client selection overhead

Table 8 reports the mean client selection duration (ST), training makespan (TT), maximum communication time (CT), and client idle time (IT) across the 100 communication rounds. Each entry is a per-round wall-clock mean in seconds, not a total over 100 rounds.

Table 8: Mean per-round client selection overhead in seconds.

Client Selection Algorithm	First Set of Experiments				Second Set of Experiments			
	ST	***TT***	***CT***	***IT***	***ST***	***TT***	***CT***	***IT***
Random	0.01	15.79	6.06	0	0.01	26.88	7.58	0
MEC	8.55	7.21	6.87	0	17.34	8.73	6.71	1.9
ECMTC	6.08	12.98	6.11	0	17.94	14.6	6.37	0
ECMTC_D15	5.28	11.53	5.79	0	16.02	12.45	5.82	0
OLAR	0.86	7.96	6.67	0	1.72	10.19	6.41	0
$(MC)^2MKP$	2.28	13.22	6.0	0	6.12	15.09	6.42	0
ELASTIC	0.82	10.0	6.63	0	1.49	15.8	6.47	0
FedAECS	0.99	10.63	5.93	0	1.54	16.12	7.44	0

In the first set of experiments, no client experienced idle time. In the second set, MEC caused an average idle time of 1.9 seconds per round, corresponding to a 12.31% overhead. Since the interpolation and extrapolation procedure enforces monotonically increasing cost estimates, the MinMaxTime step of MEC could be replaced by OLAR to reduce execution time, as OLAR is optimal for makespan under this cost behavior (8). However, for more complex models or slower communication links, this overhead may be hidden by longer training or communication times.

4.3.3.2 Overall system and model performance

Tables 9 and 10 summarize the mean number of selected clients, total training time, total energy consumption, and final test accuracy across the 100 communication rounds.

In the first set of experiments, 12,500 training tasks and 2,500 testing tasks are scheduled per round, corresponding to 25% of the total training and testing data. This reduced data availability affects the quality of models trained by algorithms that select more clients, as each selected client receives fewer local samples. Compared with Random, MEC and OLAR reduce total training time by 54.33% and 49.62%, respectively,

Table 9: Summary of metrics for the first set of experiments.

Client Selection Algorithm	Mean Number of Selected Clients	Total Time (s)	Total Energy Consumption (J)	Final Accuracy
Random	25	1,579.19	1,039,519.74	0.10
MEC	50	721.22	1,486,976.73	0.10
ECMTC	13.58	1,298.15	763,300.76	0.40
ECMTC_D15	13.62	1,152.81	767,812.95	0.38
OLAR	50	795.65	1,493,421.76	0.10
$(MC)^2MKP$	13.58	1,322.24	764,808.14	0.40
ELASTIC	50	1,000.28	1,502,038.71	0.10
FedAECS	22.93	1,062.69	953,130.32	0.10

Table 10: Summary of metrics for the second set of experiments.

Client Selection Algorithm	Mean Number of Selected Clients	Total Time (s)	Total Energy Consumption (J)	Final Accuracy
Random	25	2,687.71	1,615,671.34	0.45
MEC	50	872.96	2,030,212.43	0.10
ECMTC	25.25	1,460.16	1,522,079.18	0.41
ECMTC_D15	25.58	1,244.79	1,532,521.77	0.45
OLAR	50	1,019.37	2,051,372.70	0.10
$(MC)^2MKP$	25.25	1,508.98	1,519,950.96	0.43
ELASTIC	50	1,579.53	2,078,821.69	0.10
FedAECS	25.25	1,611.62	1,541,478.76	0.41

while ECMTC and $(MC)^2MKP$ reduce total energy consumption by 26.57% and 26.43%, respectively.

In the second set of experiments, 25,000 training tasks and 5,000 testing tasks are scheduled per round, corresponding to 50% of the total training and testing data. Compared with Random, MEC and OLAR reduce total training time by 67.52% and 62.07%, respectively, while ECMTC and $(MC)^2MKP$ reduce total energy consumption by 5.79% and 5.92%, respectively. Although ECMTC consumes slightly more energy than $(MC)^2MKP$, it requires 3.24% less training time.

4.3.3.3 Round-level training performance

The round-level analysis focuses on the second set of experiments because it employs a more representative workload. Random selection is considered as the baseline. Figure 7 shows the makespan, energy consumption, and accuracy obtained during the training phase in each communication round.

For makespan, MEC and OLAR outperform all other algorithms, whereas Random produces the worst results. On average, MEC and OLAR complete the training phase 18.15 and 16.68 seconds faster than Random, respectively. The difference between MEC

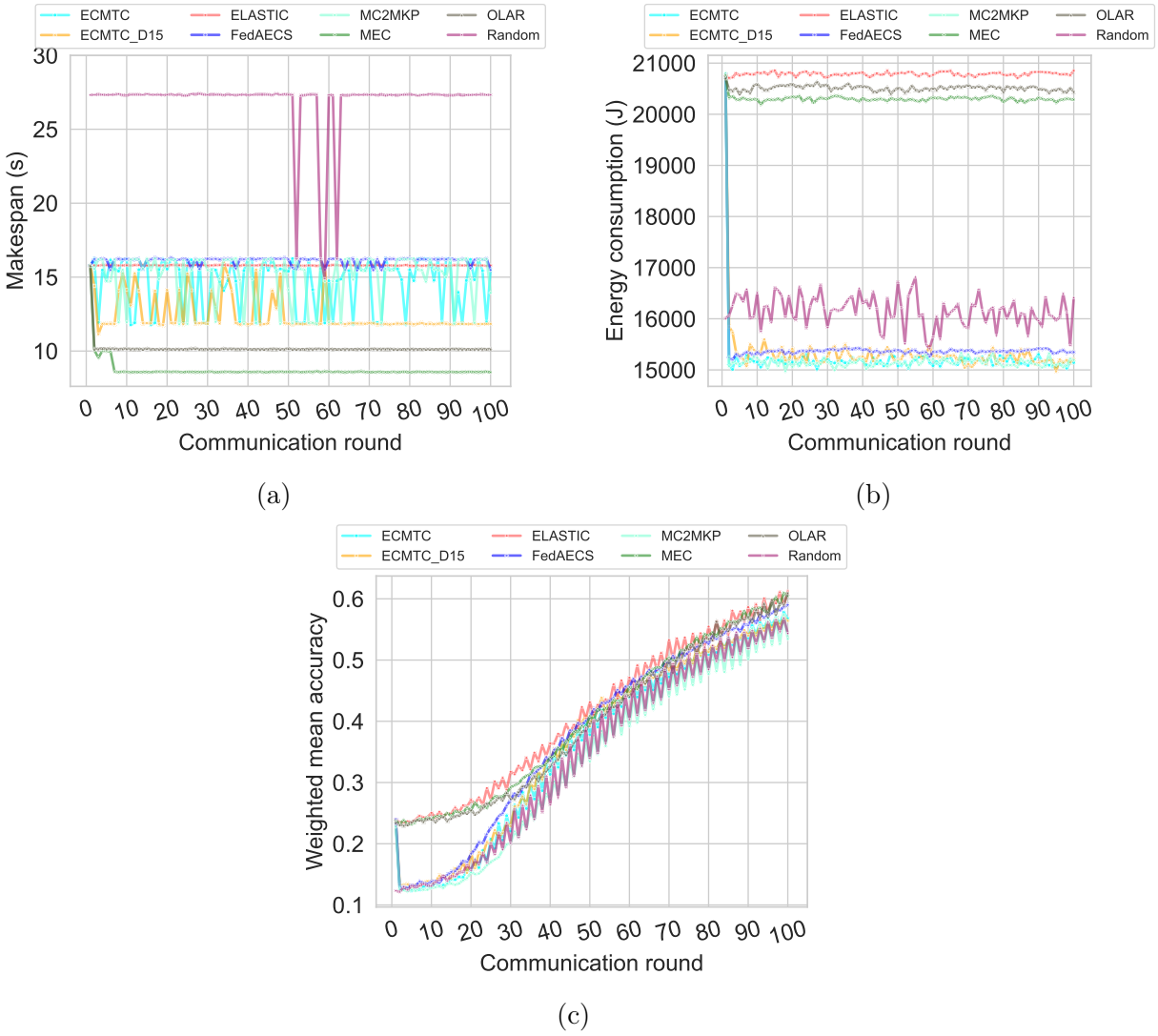

Figure 7: Results for the second set of experiments regarding makespan (a), energy consumption (b), and accuracy (c) during the training phase (25,000 tasks scheduled).

and OLAR is likely due to performance variation across client nodes during execution and requires further investigation. For ECMTC_D15, although schedules are generated with a 15-second deadline, this deadline is violated in 7 out of 100 rounds due to performance variation and estimation errors.

For energy consumption, ECMTC and $(MC)^2MKP$ outperform the remaining algorithms. On average, they consume 935.92 and 957.20 fewer joules than Random per training round, respectively. This result highlights the importance of energy-aware selection: ECMTC and $(MC)^2MKP$ select approximately the same number of clients as Random, but choose them based on efficiency.

4.3.3.4 Testing accuracy

Figure 8 shows the test accuracy achieved across communication rounds for the second set of experiments.

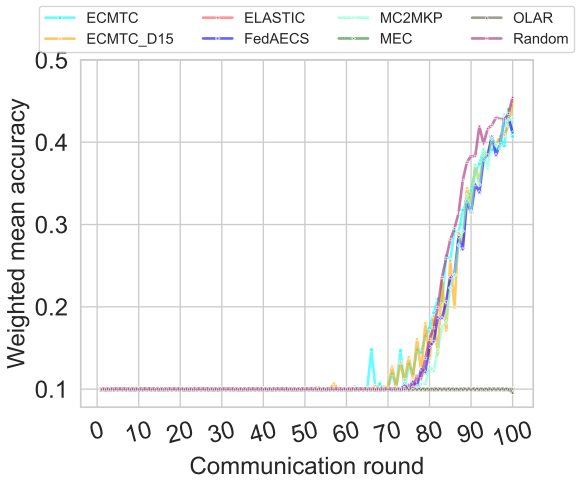

Figure 8: Results for the second set of experiments regarding accuracy during the testing phase (5,000 tasks scheduled).

MEC, OLAR, and ELASTIC achieve slightly higher training accuracy because they select all available clients in each round. However, the test accuracy curves suggest that these algorithms may suffer from overfitting. Since all clients are selected, each client trains on fewer local samples, which can reduce the effectiveness of the learned global model. For the remaining algorithms, the test accuracy curves follow similar trends, suggesting that in this balanced data scenario, client selection has a limited impact on global model quality.

4.3.3.5 Discussion on performance metrics

The results show that client selection should be aligned with the optimization objective. MEC and OLAR are more effective when the objective is to minimize training time, whereas ECMTC and $(MC)^2MKP$ are more effective when the objective is to reduce energy consumption. However, optimizing system-level metrics alone is not always sufficient to ensure model quality. In particular, when the total workload is fixed, selecting many clients can reduce the amount of data processed by each client, which may harm generalization. Therefore, model-quality indicators such as accuracy, loss, or data-distribution awareness should also be considered when clients have limited data or imbalanced local distributions.

4.3.4 General Discussion

The preliminary results show that modeling FL client selection as a task scheduling problem is useful for controlling system-level metrics. In the simulation-based experiments, MEC achieved the best makespan results, while ECMTC achieved the best energy-consumption results under deadline constraints. The real-world experiments further confirmed that these objectives can translate into practical FL executions, with MEC reducing total training time and ECMTC reducing total energy consumption compared with Random and other baseline strategies.

However, these findings are limited by the preliminary nature of the evaluation. The real-world experiments considered balanced and disjoint CIFAR-10 partitions, where clients had the same number of training and testing samples and comparable class distributions. Therefore, they did not cover non-IID scenarios, where clients may differ in data volume, class distribution, or access to minority classes. In such cases, optimizing only execution time and energy consumption may not be sufficient to preserve model quality.

MEC has limitations related to its makespan-oriented objective. Since it prioritizes the minimization of round makespan, it tends to select all or almost all available clients when this reduces the amount of work assigned to each one. Although this behavior is useful for reducing round training time, it may also cause each selected client to train on a small portion of its local data when the total workload is fixed. As observed in the real-world experiments, selecting many clients does not necessarily improve test accuracy. This suggests that minimizing makespan may slow model convergence or reduce the quality of the learned model when the workload assigned to each client becomes too small.

ECMTC may introduce selection bias because of its energy-oriented objective. Since it prioritizes energy consumption, it tends to favor the most energy-efficient clients. When client characteristics remain stable across rounds, this may lead to the repeated selection of the same subset of clients. Although this behavior is desirable from an energy-efficiency perspective, it can reduce participation diversity. In non-IID scenarios, repeatedly selecting the same energy-efficient clients may bias the global model toward their data distribution while underrepresenting other classes, data patterns, or client groups.

More generally, both MEC and ECMTC are unaware of the statistical properties of client data. They do not explicitly consider class distribution, data diversity, loss, accuracy progression, or historical participation. Consequently, they may select clients that are optimal from a system perspective but suboptimal from a learning perspective.

This limitation is particularly relevant when data are heterogeneous or imbalanced, or when minority classes are concentrated in clients that are slower or less energy efficient. In these cases, optimizing only time and energy may reduce resource consumption while producing a less robust global model.

The real-world experiments also highlight a practical modeling aspect of scheduling-based client selection. While the simulation-based experiments assume that execution time and energy consumption are known for each task assignment, real FL deployments must estimate these values from observed client behavior. In this chapter, interpolation and extrapolation are used to estimate unobserved task assignments from previous rounds. These estimates allow MEC and ECMTC to operate in practice, but they may differ from the actual costs observed during execution, especially when system load or communication conditions vary. The continual profile refresh incorporates changes observed when a client participates, but estimates for unselected clients may become stale, and abrupt contention between selection and execution cannot be anticipated. The deadline constrains predicted assignment costs rather than unforeseen in-round delays, while dropping unfinished work or aggregating partial results to enforce a hard deadline remains outside the scope of MEC and ECMTC. As observed with ECMTC_D15, such variation can lead to occasional deadline violations.

A further limitation is that MEC and ECMTC optimize measured energy consumption but do not impose a per-client battery or renewable-energy budget. A natural extension would associate each client i with an up-to-date spendable energy budget B_i and require $E_i(x_i) \leq B_i$ for every selected client. This would support, for example, minimizing makespan without exceeding current battery levels or intermittently available green-energy allocations. Such an extension would require clients to refresh their available budgets and the server to treat assignments whose predicted energy exceeds those budgets as infeasible.

Overall, these observations motivate the broader client selection strategy developed in the following chapter. While MEC and ECMTC provide optimal solutions for time- and energy-oriented objectives, they do not account for data distribution, client diversity, model utility, or fairness across rounds. Thus, this preliminary analysis highlights both the benefits and limitations of optimal scheduling-based client selection, motivating a broader multi-objective approach that jointly considers system efficiency and learning quality.

5 MetaCS-FL: A Metaheuristic-Based Framework for Client Selection

This chapter addresses client selection in FL from four complementary perspectives: execution time, energy consumption, model performance, and participation fairness. In synchronous FL, the duration of each training round is determined by the slowest selected client, commonly referred to as the straggler (8, 102, 99). At the same time, energy consumption is a critical concern in edge and mobile environments, where devices are constrained by limited battery capacity (103), variable energy availability (104), and the broader environmental impact of distributed computation (105). Beyond these system-level constraints, client selection also influences learning quality and participation fairness. Selecting too few clients may reduce the representativeness of the training data and degrade model accuracy, whereas repeatedly excluding the same clients may discourage participation (13) and weaken the collaborative nature of FL. Encouraging diverse client participation across rounds exposes the global model to a wider range of local data distributions, which may improve robustness and help mitigate bias (14, 15).

To address these challenges, this chapter presents *MetaCS-FL*, a metaheuristic-based framework for client selection in cross-device FL systems. Rather than only deciding which clients participate, MetaCS-FL formulates client selection as a task scheduling problem, in which the server also determines how much data each selected client processes locally. The framework incorporates reliability-aware workload allocation by assigning each client a score based on availability and task completion history. This score constrains the workload of less reliable clients, allowing them to remain eligible while reducing the risk of unfinished work. MetaCS-FL supports different metaheuristics, initial solution methods, and user-defined triggers for new selections. It uses client profiling, current availability, and historical performance information to search for schedules that balance makespan, energy consumption, participation diversity, class-distribution quality, and historical loss-based utility. In this work, MetaCS-FL uses the optimal schedule generated by *ECMTC* (28) as its initial solution and refines it through Large Neighborhood Search (LNS) (72). Section 5.1 formalizes the system model and multi-objective client selection problem. Section 5.2 introduces the MetaCS-FL framework and its selection procedure.

5.1 System Model and Problem Definition

This section establishes the modeling basis for MetaCS-FL. It defines the assumptions and system behavior of the considered synchronous cross-device FL setting, then formulates client selection as a multi-objective task scheduling problem involving time, energy, participation diversity, local data distribution, expected utility, and reliability.

5.1.1 General Assumptions

The system model is based on the following general assumptions:

1. The FL execution is synchronous, coordinated by a central server, and consists of two main phases: *Initialization* and *FL Loop*;
2. The server is trusted but may request hardware characteristics, performance metrics, and local data statistics for client selection;
3. Model evaluation is performed on the clients to assess the resulting model utility within each local context;
4. Clients are assumed to be honest; therefore, the system does not account for malicious behavior or attacks;
5. Clients may join, leave, or become temporarily unavailable during the FL execution;
6. The server monitors each client's availability and completion status, updating individual reliability over time.

5.1.2 System Model

Based on the general assumptions defined above, Figures 9 and 10 illustrate the sequence of events in the considered synchronous cross-device FL system.

5.1.2.1 Initialization

In the *Initialization* phase (Fig. 9), the *Server* first waits until a predefined minimum number of *Clients* have established a connection (①). If no historical performance records are available at the beginning of a new execution, the server performs a synchronous profiling step with the initial clients. In this step, each initial client uses a small portion

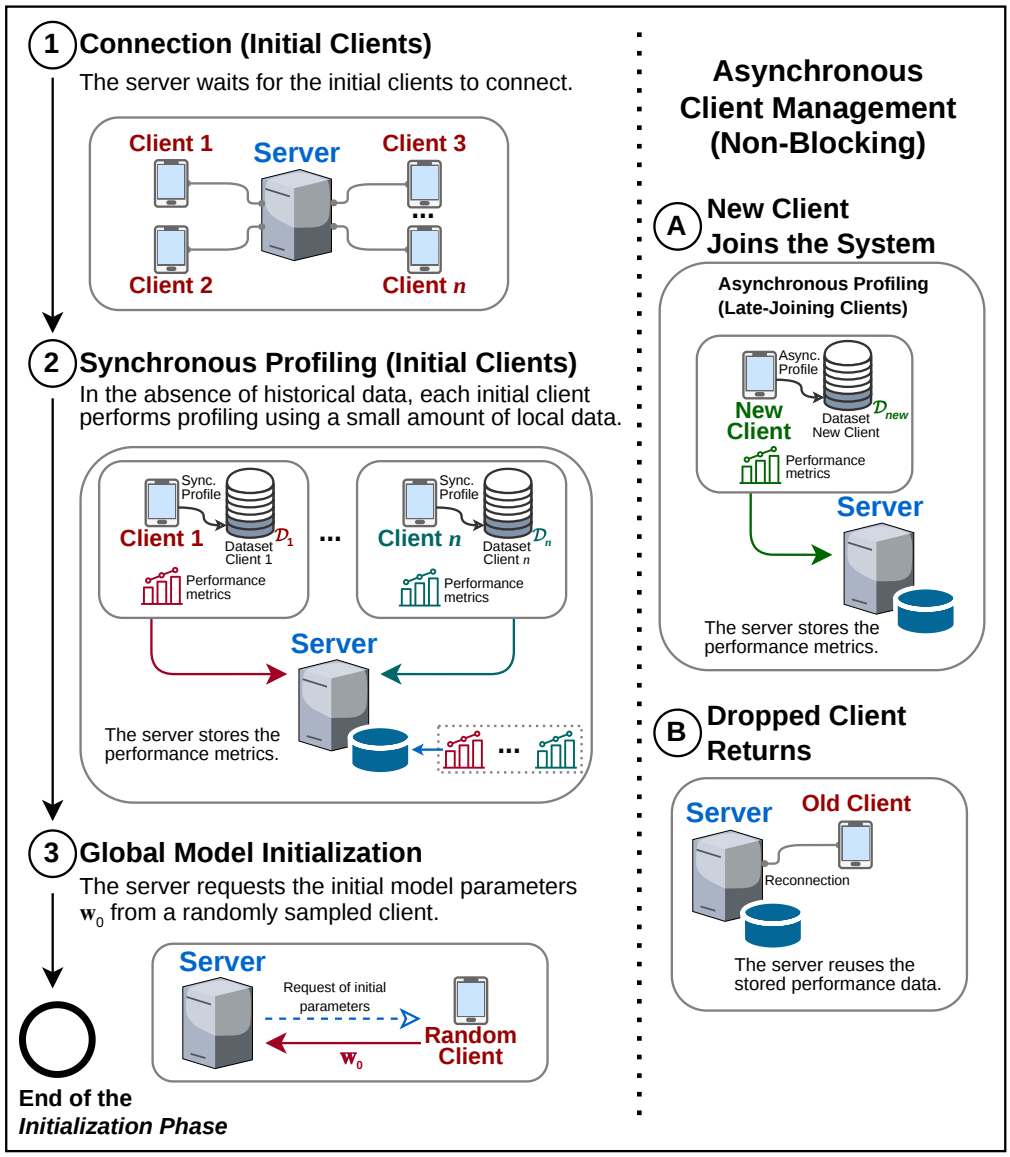

Figure 9: The initialization phase.

of its local data to estimate execution- and system-level characteristics, such as processing time, energy consumption, and communication quality. The resulting cold-start profiles are returned to the server, stored as initial performance records, and used as fallback information for client selection whenever no more recent observations are available (②). After the synchronous profiling step is completed, the server requests the initial model parameters, denoted by $\mathbf{w}_0$ in Fig. 9, from a randomly sampled client and uses them to initialize the global model (③). These initialized parameters will then be distributed to the selected clients for training at the start of the first FL round.

The server also handles dynamic client availability through an asynchronous client-management procedure. Clients that join after the initial connection stage, referred to

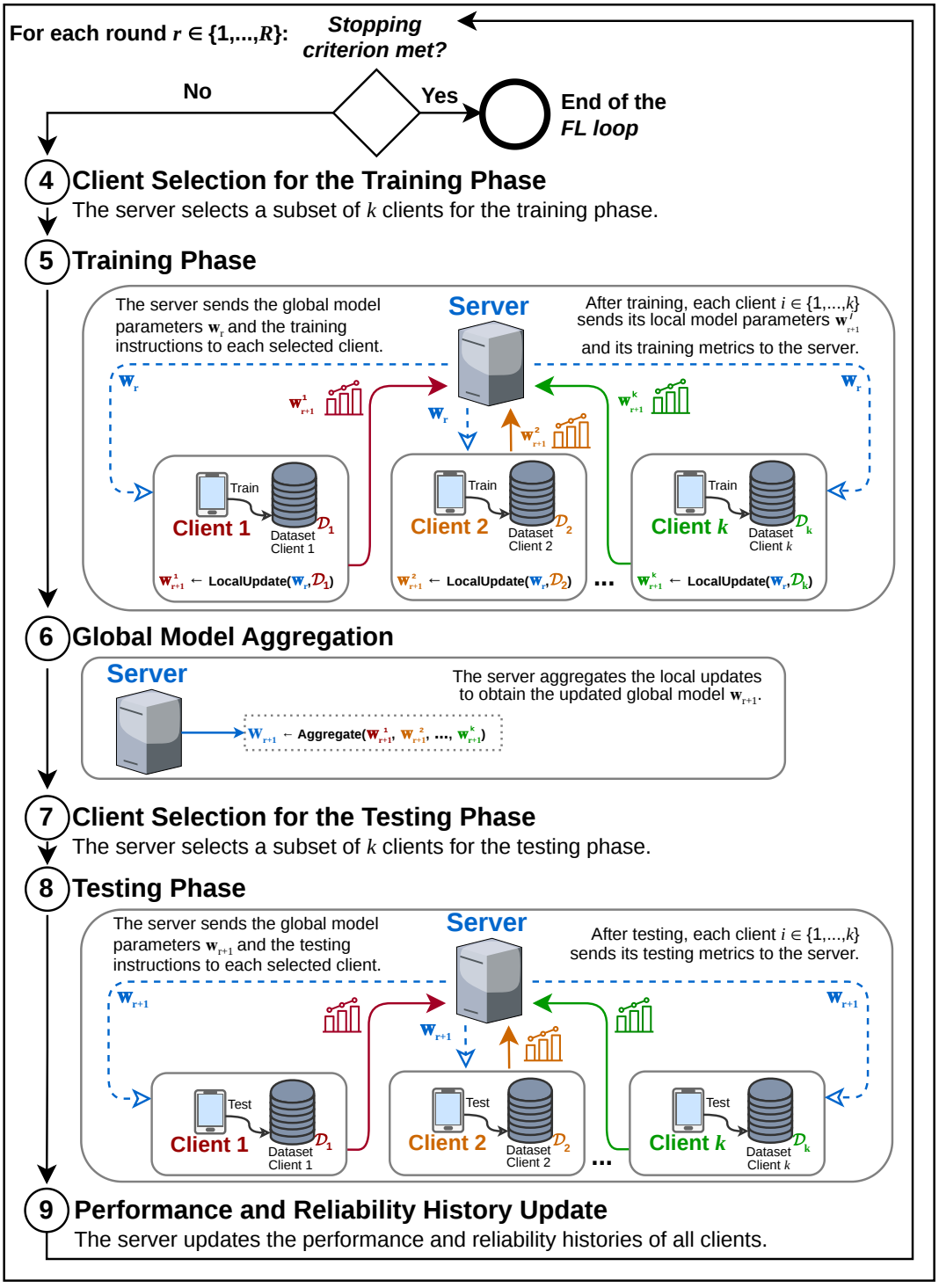

Figure 10: The federated learning loop.

hereafter as late-joining clients, are profiled asynchronously while the FL process continues, allowing training to proceed without waiting for their profiling results (Ⓐ). Once their profiling metrics become available, they are added to the effective candidate set for subsequent rounds. Returning clients, such as those that temporarily disconnect due to network instability, battery depletion, or other availability fluctuations, do not repeat pro-

filing if a valid profile is already stored; instead, the server reuses the stored performance data unless a profile refresh is required (Ⓑ).

5.1.2.2 FL loop

The *Training* and *Testing* phases (Fig. 10) constitute the core of the Federated Learning loop. For each communication round $r \in 1,\ldots,R$, the server first verifies whether the stopping criterion has been met. If the criterion is satisfied, the FL loop terminates; otherwise, the server proceeds with a new round.

In the training phase, the server selects a subset of k clients based on their available profiles and performance records (④). It then sends the current global model parameters $\mathbf{w} * r$ and the training instructions to the selected clients $i \in 1,\ldots,k$. Each client i trains the received model using its local dataset $\mathcal{D} * i$, producing local model parameters $\mathbf{w} * r + 1^i$. After training, each client i sends its local model parameters $\mathbf{w} * r + 1^i$ and the corresponding training metrics back to the server (⑤).

The server aggregates the received local model parameters $\mathbf{w} * r + 1^i$ to obtain the updated global model parameters $\mathbf{w} * r + 1$ (⑥). After aggregation, the server selects a subset of k clients for the testing phase (⑦). The server sends the updated global model parameters $\mathbf{w}_{r+1}$ and the testing instructions to the selected clients $i \in 1,\ldots,k$. Each client i evaluates the received model using its local data and returns the corresponding testing metrics to the server (⑧).

Finally, the server updates the performance and reliability histories of all clients using the information collected during the round, including training metrics, testing metrics, and completion status (⑨). For each training or testing phase, results are accepted only until the corresponding deadline τ, if one is specified. Results returned after τ are discarded from the current aggregation or testing record, while clients that fail to return a valid result before the deadline are recorded as incomplete and receive a completion penalty, which may reduce their reliability score and restrict admissible task capacities in subsequent selections. After each successful execution, the latest phase metrics refresh the corresponding client profile, including computation time and energy as well as communication measurements used to derive current bandwidth and latency estimates. Initial profiling therefore serves as a cold-start fallback and is not treated as an immutable profile for the entire FL execution. These histories guide client selection in subsequent rounds.

5.1.2.3 Overhead assumptions

Not all events shown in Figures 9 and 10 are modeled in the execution time and energy consumption formulation (Section 5.1.2.4). During *Initialization*, the *Connection* and *Global Model Initialization* events (① and ③ in Fig. 9) are assumed to introduce negligible overhead. In contrast, the *Synchronous Profiling* event (② in Fig. 9) is explicitly considered, since the server blocks until the initial clients complete profiling using a small amount of local data. Thus, the profiling time is determined by the slowest initial client.

Late-joining clients are handled through *Asynchronous Profiling* as part of *Asynchronous Client Management* (Ⓐ in Fig. 9). Therefore, their profiling overhead does not block the ongoing FL round and is not added to the critical-path execution time of the current training or testing phase. Once their profiling metrics become available, they are used in subsequent client selection decisions. Returning clients that already have a valid stored profile reconnect without repeating profiling unless a profile refresh is required (Ⓑ in Fig. 9); consequently, profile reuse is assumed to introduce negligible overhead.

During the FL loop, the *Training Phase* and *Testing Phase* (⑤ and ⑧ in Fig. 10) are explicitly modeled. For each selected client, the time spent in either phase is represented as the sum of download, computation, and upload times, while the corresponding energy consumption is computed from the time spent in each component and the associated mean power consumption. Therefore, these phases capture both client-side computation and client-server communication costs.

The overhead of *Client Selection* (④ and ⑦ in Fig. 10) depends on the computational complexity of the adopted selection algorithm and is discussed in Section 5.2. By contrast, *Global Model Aggregation* (⑥) is assumed to introduce negligible overhead, as it is performed by the server, which is expected to have substantially greater computational capacity than the clients. Similarly, the *Performance and Reliability History Update* (⑨) is treated as a lightweight server-side bookkeeping operation and is not explicitly included in the execution time or energy consumption formulation.

5.1.2.4 Modeling of execution time and energy consumption

The **time** spent by a selected client i during the **training** or **testing** phase of round r (⑤ and ⑧ in Fig. 10), as defined in Eq. (5.1), is computed as the sum of the following components: (i) **download time** ($\mathrm{T}^{D}_{\mathrm{phase}}$), corresponding to the time required to download the global model parameters and the associated training or testing instructions from the

server; (ii) **computation time** ($\mathrm{T}^{C}_{\mathrm{phase}}$), representing the time spent performing local training or testing; and (iii) **upload time** ($\mathrm{T}^{U}_{\mathrm{phase}}$), which, during training, accounts for uploading local model updates and training metrics to the server, and during testing, accounts for uploading the testing metrics.

$$\mathrm{T}_{\mathrm{phase}} \coloneqq \mathrm{T}^{D}_{\mathrm{phase}} + \mathrm{T}^{C}_{\mathrm{phase}} + \mathrm{T}^{U}_{\mathrm{phase}} \tag{5.1}$$

The **energy** consumed by a selected client i during the **training** or **testing** phase of round r (⑤ and ⑧ in Fig. 10) is modeled using a standard linear relationship between time and power. Accordingly, the total energy consumption is given by:

$$\mathrm{E}_{\mathrm{phase}} \coloneqq \mathrm{E}^{D}_{\mathrm{phase}} + \mathrm{E}^{C}_{\mathrm{phase}} + \mathrm{E}^{U}_{\mathrm{phase}} \tag{5.2}$$

where each component is computed as $\mathrm{E}^{X}_{\mathrm{phase}} \coloneqq \mathrm{T}^{X}_{\mathrm{phase}} \cdot p\text{-}\mathrm{X}_i,\ X \in \{D, C, U\}$, and $p\text{-}\mathrm{X}_i$ denotes the mean power consumption of client i during phase component X (*i.e.*, download, computation, or upload). At selection time, the server recomputes both communication and computation components using the latest available client information: communication from the reported download bandwidth, upload bandwidth, and latency, and computation from the most recent phase record or, when no phase history exists, the closest profiling record, scaled to each candidate workload. Consequently, both components are updated at runtime instead of remaining fixed at their initialization values.

5.1.2.5 Privacy-preserving disclosure of client data distributions

Privacy is a key concern in FL systems, as clients collaboratively train a global model without sharing raw data. However, even when raw data remain local, information may still be inferred from model updates or auxiliary metadata. As stated in the system assumptions, clients are considered honest and are not modeled as malicious adversaries; therefore, this work does not address attacks based on falsified statistics, manipulated updates, or other forms of Byzantine behavior. Many existing works address privacy leakage, for instance, by adding noise to clients' local parameters or gradients before aggregation (27). In contrast, within this honest-client setting, our approach applies *Differential Privacy* (DP) directly to clients' local class distributions, generating noisy histograms once when clients connect to the system.

Applying DP at the data-distribution level is well suited to the proposed client selec-

tion mechanism because the resulting statistics can be cached and reused across rounds without repeatedly querying clients. This design is particularly relevant in cross-device FL, where clients may be intermittently available and where additional coordination can increase communication overhead. Cryptographic approaches such as *Secure Aggregation* (45) provide strong protection for aggregate values, but they rely on interactive multi-party protocols in which each client's contribution is coupled to the participation of others. Although such protocols are effective for protecting training updates during aggregation, their round-dependent and interactive nature is less aligned with the one-time disclosure and reuse of per-client auxiliary statistics, such as local class distributions.

The server knows the target model and the allowed data types, including the set of global classes. Upon connection, each client is queried once for its local class distribution in a privacy-preserving manner. The server provides the global class index map, the privacy parameter ϵ, which controls the privacy strength, and the total number of global classes K. For each requested data split $s \in \{\text{train}, \text{test}\}$, client i maps its local labels to global class indices and constructs a local histogram. Let $\mathcal{D}_i^s$ denote the local data split of client i, and let $g(y)$ be the mapping from a local label y to its corresponding global class index. The count associated with global class k is given by

$$h_{i,k}^s = \sum_{(x,y)\in\mathcal{D}_i^s} \mathbb{1}\{g(y) = k\}, \quad k \in \{0, \ldots, K-1\} \tag{5.3}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function. The complete local histogram for split s is therefore

$$\mathbf{h}_i^s = \left[h_{i,0}^s, h_{i,1}^s, \ldots, h_{i,K-1}^s\right] \tag{5.4}$$

To protect the exact class distribution, each client perturbs the histogram locally using the Laplace mechanism (44, 110). Since each record contributes to exactly one histogram bin, the ℓ_1 sensitivity of the histogram query is one under add/remove adjacency. Thus, independent Laplace noise is added to each class count:

$$\bar{h}_{i,k}^s = h_{i,k}^s + \eta_{i,k}^s, \quad \eta_{i,k}^s \sim \text{Laplace}\left(0, \frac{1}{\epsilon}\right) \tag{5.5}$$

More generally, the Laplace mechanism calibrates the noise scale as $b = \Delta_1/\epsilon$, where Δ_1 is the ℓ_1 sensitivity of the query. Since $\Delta_1 = 1$ for the histogram under add/remove adjacency, the scale used here is $b = 1/\epsilon$. A random variable $\eta \sim \text{Laplace}(0,b)$ has

probability density

$$f_\eta(z) = \frac{1}{2b} \exp\left(-\frac{|z|}{b}\right) = \frac{\epsilon}{2} \exp\left(-\epsilon|z|\right), \quad z \in \mathbb{R} \tag{5.6}$$

This distribution is symmetric around zero, with expected value $\mathbb{E}[\eta] = 0$, variance $\mathrm{Var}(\eta) = 2/\epsilon^2$, and expected absolute perturbation $\mathbb{E}[|\eta|] = 1/\epsilon$. Its tail probability is $\Pr(|\eta| \geq a) = \exp(-\epsilon a)$ for $a \geq 0$. Consequently, decreasing ϵ produces a wider distribution and stronger privacy at the cost of less accurate counts, whereas increasing ϵ concentrates the noise around zero and improves histogram fidelity while providing weaker privacy. The selected scale also ensures that the likelihood ratio of any disclosed output under two adjacent local datasets is bounded by $\exp(\epsilon)$, thereby satisfying ϵ-DP.

The noisy counts are then clipped to non-negative values and rounded to integers before being sent to the server:

$$\widetilde{h}^s_{i,k} = \left\lfloor \max\left(0, \bar{h}^s_{i,k}\right) \right\rceil \tag{5.7}$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

Accordingly, the privacy-preserving histogram disclosed by client i for split s is

$$\widetilde{\mathbf{h}}^s_i = \left[\widetilde{h}^s_{i,0}, \widetilde{h}^s_{i,1}, \ldots, \widetilde{h}^s_{i,K-1}\right] \tag{5.8}$$

Since clipping and rounding are applied after noise addition, they are post-processing operations and do not weaken the DP guarantee. However, Laplace perturbation may cause classes with zero local samples to appear with positive noisy counts. Thus, the disclosed histogram may report nonzero counts for classes that are absent from the client's local data, which is an expected consequence of the privacy mechanism.

In our approach, this mechanism is applied separately to the local training and testing distributions. The server caches the resulting noisy histograms and uses them in subsequent client selection decisions without re-querying clients or exposing the exact local class distributions. These histograms should therefore be interpreted as approximate auxiliary statistics rather than exact representations of local data, especially for small ϵ values or clients with few samples in some classes.

5.1.3 Problem Definition

Let $\mathcal{C}_r$ denote the set of n clients eligible for training or testing in round r, and let t represent the configured workload, measured in data slices. An optional deadline τ may also be specified, defining the maximum time allowed for processing the assigned workload. A client is considered eligible only if it is both available and has a valid profile. Thus, clients that are unavailable, have not yet completed profiling, or require a profile refresh are excluded from $\mathcal{C}_r$.

The workload t, referred to as *tasks*, is assigned to a selected subset of clients $\mathcal{S}_r \subseteq \mathcal{C}_r$. Each task corresponds to a configured, atomic scheduling unit representing a fixed number of local samples and used to form mini-batches for local training or testing. Each client $i \in \mathcal{C}_r$ has a set of task capacities $\mathtt{AC}_i$, specifying the sizes of data slices it can handle, which are primarily determined by its number of local samples and task granularity.

A common task size specifies only the quantity of local data to process and does not imply that tasks contain statistically identical or IID samples. The decision variable x_i defines the total workload assigned to client i. For class-aware selection, this total is further represented by class-specific counts derived from the client's DP-perturbed histogram, and the corresponding class allocation is included in the local instructions. The client selects the requested records from its own dataset, so neither raw examples nor data ownership are transferred between clients or to the server.

Each client i also has a round-dependent reliability-adjusted capacity set $\widetilde{\mathtt{AC}}_{i,r}$, which specifies the data-slice sizes it can handle in round r after accounting for its updated reliability score (Section 5.1.3.1). Since this set includes 0 to represent non-selection, each client may be assigned $x_i \in \widetilde{\mathtt{AC}}_{i,r}$ tasks, with $x_i = 0$ indicating that client i is not selected.

Table 11 summarizes the notation used in the optimization problem definition.

5.1.3.1 Reliability score

Let $rs_i \in [0,1]$ denote the reliability score of client i, maintained by the server for each client that has joined the system. When a client first joins, its reliability score is initialized as 1, meaning that no reliability-based restriction is initially imposed. The score is computed from the client's previous availability and completion behavior and is used to proportionally restrict its effective task assignment capacities before solving the scheduling problem. The goal is to mitigate the risk of client dropout or incomplete processing by reducing the workload assigned to clients with less reliable past behavior.

Table 11: Notation summary for the optimization problem.

Symbol	Meaning
r	Current FL round.
$\mathcal{C}_r$	Set of available clients in round r.
$\mathtt{AC}_i$	Nominal task capacities of client $i \in \mathcal{C}_r$.
$\widetilde{\mathtt{AC}}_{i,r}$	Reliability-aware task capacities of client i in round r.
t	Total workload size (user-defined).
$\mathcal{S}_r$	Set of selected clients in round r.
τ	Time limit for training/testing (user-defined).
x_i	Tasks assigned to client i in round r.
$x_i^{(j)}$	Tasks assigned to client i at the j-th most recent participation.
rs_i	Reliability score of client i.
$\overline{rs}_i$	Previous reliability score of client i.
$\lambda_{i,r-1}$	Reliability penalty assigned to client i based on round $r-1$.
λ_{avail}	Availability penalty for a joined client that is unavailable.
λ_{compl}	Completion penalty for a selected client that does not complete its processing.
μ	Memory weight used to compute the reliability score.
δ_i	Time taken by client $i \in \mathcal{C}_r$ to complete tasks.
M_r	Makespan: time of the slowest client.
ϵ_i	Energy consumed by client $i \in \mathcal{C}_r$.
Σ_r	Total energy consumed by all clients.
ρ_i	Participations of client i over last q rounds.
d_i	Entropy-based diversity contribution.
D_r	Client diversity score for the round.
Γ	Set of task class indices across $\mathcal{C}_r$.
$\mathtt{Y}_i$	Local task counts per class for client $i \in \mathcal{C}_r$.
$\mathrm{K}_{\mathcal{C}}$	Available tasks per class.
$\mathrm{K}_{\mathcal{S}}$	Assigned tasks per class.
K_{Cov}	Class coverage term.
K_{Imb}	Class imbalance term.
K_r	Class-distribution score.
β	Coverage-imbalance trade-off.
q_i	Recent participations considered.
$\widetilde{L}_{i,\text{train}}^{(j)}$	Normalized training loss at recent participation j.
$\widetilde{L}_{i,\text{test}}^{(j)}$	Normalized test loss at recent participation j.
ϕ_i	Learning behavior term.
ψ_i	Generalization term.
u_i	Client utility in round r.
U_r	Utility score for the round.
α	Learning-generalization trade-off.
$\mathcal{W}$	Set of weights for each objective (user-defined).
$\mathcal{X}$	Schedule assigning all tasks in the workload.
$F(\mathcal{X})$	Cost function value for schedule $\mathcal{X}$.
$\mathcal{X}^*$	Optimal schedule for the given weight set $\mathcal{W}$.

Before the selection step of round r, the server computes rs_i using the behavior observed in the previous round. For each client, the penalty $\lambda_{i,r-1}$ is defined as:

$$\lambda_{i,r-1} \coloneqq \begin{cases} 0, & \text{if } i \in \mathcal{C}_{r-1} \text{ and } i \notin \mathcal{S}_{r-1}, \\ \lambda_{\text{avail}}, & \text{if } i \notin \mathcal{C}_{r-1}, \\ 0, & \text{if } i \in \mathcal{S}_{r-1} \text{ and completed the processing}, \\ \lambda_{\text{compl}}, & \text{if } i \in \mathcal{S}_{r-1} \text{ and did not complete the processing.} \end{cases} \tag{5.9}$$

The cases in Eq. (5.9) distinguish between availability and completion outcomes. No penalty is applied when a client is available but not selected, or when a selected client completes its processing. A client that becomes unavailable after entering the system receives an availability penalty λ_{avail}, whereas a selected client that fails to complete its processing receives a completion penalty λ_{compl}. Clients that have not yet joined the system are not penalized; their reliability score is initialized upon their first connection.

The penalty values are user-defined magnitudes satisfying $0 \leq \lambda_{\text{avail}} \leq \lambda_{\text{compl}} \leq 1$. They are not required to sum to one, since they are not complementary probabilities; instead, they independently control how strongly each type of undesired behavior affects the reliability score. The condition $\lambda_{\text{avail}} \leq \lambda_{\text{compl}}$ reflects that becoming unavailable before selection is less harmful than being selected and failing to complete the processing, since the latter may directly affect the round outcome.

The reliability score used in round r is computed using an exponentially weighted moving average over the previous score and the most recent reliability observation:

$$rs_i \coloneqq \mu \cdot \overline{rs}_i + (1 - \mu) \cdot (1 - \lambda_{i,r-1}) \tag{5.10}$$

where $\overline{rs}_i$ is the previous reliability score of client i, and $\mu \in [0,1]$ is a memory weight that controls how much past behavior is preserved. Higher values of μ preserve longer-term history, while lower values make the score more sensitive to the most recent observation. The term $1 - \lambda_{i,r-1}$ represents the most recent reliability observation: it is closer to 1 when the client remains available or completes its processing, and lower when the client becomes unavailable or fails to complete its processing. As a result, the reliability score can increase when the client remains available or successfully completes its processing, and decrease under repeated unavailability or failed completion. Since Eq. (5.10) is a convex combination of values in [0,1], the score remains bounded in [0,1].

The server then uses the current reliability score rs_i to adjust the task capacity set of each client. Let $\texttt{AC}_i = [a_{i,0}, a_{i,1}, \ldots, a_{i,|\texttt{AC}_i|-1}]$ denote the ordered list of valid task capacities of client i. The reliability-aware task capacity set in round r is defined as:

$$\widetilde{\texttt{AC}}_{i,r} \coloneqq \{a_{i,j} \in \texttt{AC}_i : 0 \leq j < \max\left(1, \lfloor rs_i \cdot |\texttt{AC}_i| \rfloor\right)\} \tag{5.11}$$

This definition proportionally reduces the number of admissible capacity slots according to the client's reliability score while preserving only values that already belong to $\texttt{AC}_i$. Thus, less reliable clients remain eligible for selection, but with fewer possible workload

assignments. Since the constraint $\sum_{i\in\mathcal{C}_r} x_i = t$ remains active, workload units removed from the admissible assignments of a less reliable client must be reassigned to capacities of other clients. This redistributes processing quantity, not private data: every client still processes samples from its local dataset. The score is a retrospective control signal derived from observed availability and completion, not a probabilistic guarantee that a client will remain available. If the restricted capacities are insufficient to satisfy the workload t, the restriction can be relaxed by temporarily adding valid capacity slots from the nominal sets of the least penalized available clients until a feasible schedule exists.

The current formulation schedules exactly the configured workload t and does not create redundant copies of tasks. An alternative reliability mechanism would over-assign workload to backup clients and accept only the results completed before the deadline. Such speculative execution could reduce the effect of post-selection failures, but it would require duplicate-aware aggregation and an explicit trade-off among additional computation, communication, energy, and learning utility, and is outside the present formulation.

5.1.3.2 Makespan and total energy consumption

Let δ_i and ϵ_i denote the time and energy consumed by client i to process its assigned tasks in the current phase of round r, as defined in Eqs. (5.12) and (5.13). The **makespan** M_r and **total energy consumption** Σ_r correspond, respectively, to the processing time of the slowest client and the sum of energy consumed across all clients, as expressed in Eqs. (5.14) and (5.15).

$$\delta_i \coloneqq \mathrm{T}_{\text{phase}}(x_i) \quad \text{if} \quad x_i \neq 0, \text{ else } 0 \tag{5.12}$$

$$\epsilon_i \coloneqq \mathrm{E}_{\text{phase}}(x_i) \quad \text{if} \quad x_i \neq 0, \text{ else } 0 \tag{5.13}$$

$$\mathrm{M}_r \coloneqq \max_{i\in\mathcal{C}_r} \delta_i \tag{5.14}$$

$$\Sigma_r \coloneqq \sum_{i\in\mathcal{C}_r} \epsilon_i \tag{5.15}$$

5.1.3.3 Diversity score

Let ρ_i denote the number of times client $i \in \mathcal{C}_r$ has participated over the last q rounds, including potential participation in round r, *i.e.*, $\rho_i \coloneqq \sum_{\ell=\max(1,r-q+1)}^{r} \mathbb{I}(i \in \mathcal{S}_\ell)$. Let d_i denote the entropy-based term associated with client i, as defined in Eq. (5.16). Based on the Shannon evenness index (111), the **client diversity score** D_r quantifies the uniformity of client participation, and is defined in Eq. (5.17).

$$d_i \coloneqq \begin{cases} -\dfrac{\rho_i}{\sum_{j\in\mathcal{C}_r}\rho_j}\ln\left(\dfrac{\rho_i}{\sum_{j\in\mathcal{C}_r}\rho_j}\right), & \text{if } \rho_i > 0, \\ 0, & \text{if } \rho_i = 0. \end{cases} \tag{5.16}$$

$$\mathrm{D}_r \coloneqq \frac{\sum_{i\in\mathcal{C}_r} d_i}{\ln(|\mathcal{C}_r|)} \tag{5.17}$$

5.1.3.4 Class-distribution score

Let $\Gamma = \{1,\ldots,m\}$ denote the set of task class indices observed among the available clients $\mathcal{C}_r$. Each client $i \in \mathcal{C}_r$ has a vector of local task counts per class, $\mathtt{Y}_i = \{\upsilon_1,\ldots,\upsilon_m\}$, where $\upsilon_k = 0$ if class k is absent from i's data. Let $\mathrm{K}_{\mathcal{C}}$ and $\mathrm{K}_{\mathcal{S}}$ represent the vectors of available tasks per class across all available clients and of assigned tasks per class across the selected clients, respectively. For a candidate schedule, each total assignment x_i is decomposed into class-specific task counts according to $\mathtt{Y}_i$ and the adopted local allocation rule. Aggregating these counts across selected clients produces $\mathrm{K}_{\mathcal{S}}$. Hence, the server evaluates not only how many samples each client will process, but also the estimated class composition of that workload.

The **class-distribution score** K_r is composed of two complementary components: a class coverage term K_{Cov}, which measures how much of the available data per class is utilized, and a class imbalance term K_{Imb}, which measures how far the assigned tasks deviate from a perfectly balanced distribution. These components are defined in Eqs. (5.18) and (5.19).

$$\mathrm{K}_{Cov} \coloneqq \frac{1}{m}\cdot\sum_{k\in\Gamma}\frac{\mathrm{K}_{\mathcal{S}}[k]}{\mathrm{K}_{\mathcal{C}}[k]} \tag{5.18}$$

$$\mathrm{K}_{Imb} \coloneqq \frac{\sqrt{\frac{1}{m}\cdot\sum_{k\in\Gamma}\left(\mathrm{K}_{\mathcal{S}}[k]-\frac{t}{m}\right)^2}}{\frac{t}{\sqrt{m}}} \tag{5.19}$$

Thus, the class-distribution score K_r is defined in Eq. (5.20):

$$\mathrm{K}_r \coloneqq \beta\cdot\mathrm{K}_{Cov} + (1-\beta)\cdot(1-\mathrm{K}_{Imb}) \tag{5.20}$$

where the parameter $\beta \in [0,1]$ controls the relative importance of K_{Cov} and K_{Imb} terms.

5.1.3.5 Utility score

Let q_i denote the number of most recent participations of client i considered prior to round r, *i.e.*, $q_i \coloneqq \min\big(q, \sum_{\ell=1}^{r-1} \mathbb{I}(i \in \mathcal{S}_\ell)\big)$. Let $x_i^{(j)}$ denote the number of tasks assigned to client i at the j-th most recent participation. Let $\widetilde{L}_{i,\text{train}}^{(j)}$ and $\widetilde{L}_{i,\text{test}}^{(j)}$ denote the normalized training and test losses at that participation, respectively, derived from historical observations, normalized to [0,1], and available prior to client selection. The terms ϕ_i and ψ_i capture the learning behavior and generalization performance of client i over time, respectively, and are defined in Eqs. (5.21) and (5.22):

$$\phi_i \coloneqq 1 - \frac{\sum_{j=1}^{q_i} x_i^{(j)} \cdot \widetilde{L}_{i,\text{train}}^{(j)}}{\sum_{j=1}^{q_i} x_i^{(j)}} \tag{5.21}$$

$$\psi_i \coloneqq 1 - \frac{\sum_{j=1}^{q_i} x_i^{(j)} \cdot \widetilde{L}_{i,\text{test}}^{(j)}}{\sum_{j=1}^{q_i} x_i^{(j)}} \tag{5.22}$$

Let u_i denote the utility of client i in round r, combining ϕ_i and ψ_i through a weighted aggregation, where x_i is its assigned workload in the current round. The utility is defined in Eq. (5.23):

$$u_i \coloneqq \frac{x_i}{t} \cdot (\alpha \cdot \phi_i + (1-\alpha) \cdot \psi_i) \quad \text{if} \quad x_i \neq 0, \text{ else } 0 \tag{5.23}$$

where the parameter $\alpha \in [0,1]$ controls the relative importance of the ϕ_i and ψ_i terms.

The **utility score** U_r captures the expected contribution to the learning process in round r, and is defined in Eq. (5.24):

$$\mathrm{U}_r \coloneqq \sum_{i \in \mathcal{C}_r} u_i \tag{5.24}$$

5.1.3.6 Optimization problem

Let $\mathcal{X} = (x_1, \ldots, x_n)$ denote a schedule that assigns $x_i \in \widetilde{\texttt{AC}}_{i,r}$ tasks to each client $i \in \mathcal{C}_r$. Let $F(\mathcal{X})$ denote a weighted scalarization of multiple objective components, where each weight w_j, $j \in \{1, \ldots, 5\}$, reflects the relative importance of the corresponding component (112). The objective function $F(\mathcal{X})$ is given in Eq. (5.25).

$$
\begin{aligned}
\textbf{min} \quad & F(\mathcal{X}) = (w_1 \cdot \mathrm{M}_r) + (w_2 \cdot \Sigma_r) - (w_3 \cdot \mathrm{D}_r) \\
& \quad - (w_4 \cdot \mathrm{K}_r) - (w_5 \cdot \mathrm{U}_r) \\
\textbf{subject to} \quad & \mathrm{M}_r \leq \tau, && \mathrm{M}_r, \tau \in \mathbb{R}_{\geq 0} \\
& \sum_{i \in \mathcal{C}_r} x_i = t, && x_i, t \in \mathbb{N} \\
& x_i \in \widetilde{\mathtt{AC}}_{i,r}, && \forall i \in \mathcal{C}_r
\end{aligned} \tag{5.24}
$$

The values of M_r and Σ_r are normalized separately to the range [0,1] using log-scaled min-max normalization with bounds derived from prior observations. This normalization removes differences in physical units, such as seconds and joules, while reducing the influence of large variations within each metric. In this scaling, 1 corresponds to the highest cost and 0 to the lowest, ensuring compatibility with the other components of $F(\mathcal{X})$.

Scalarization supplies the optimization process with one score while exposing application priorities through the set of objective weights. For interpretability, the weight sets used in this work are non-negative and sum to one, so their relative values express the intended priorities. The fixed weights used in the experiments provide a controlled comparison and place more emphasis on the learning- and participation-oriented terms that are absent from the ECMTC initial solution, while adaptive weighting and objective-ablation studies remain extensions needed to quantify interactions among all terms and determine whether smaller objective subsets suffice in particular deployments.

Let $\mathcal{W} = \{w_1, \ldots, w_5\}$, $\mathcal{A} = \{\widetilde{\mathtt{AC}}_{1,r}, \ldots, \widetilde{\mathtt{AC}}_{n,r}\}$, $\mathcal{Y} = \{\mathtt{Y}_1, \ldots, \mathtt{Y}_n\}$, $\mathcal{G} = \{\mathtt{T}_1, \ldots, \mathtt{T}_n\}$, and $\mathcal{E} = \{\mathtt{E}_1, \ldots, \mathtt{E}_n\}$ denote, respectively, the sets of objective weights, reliability-adjusted task assignment capacities, vectors of local task counts per class, estimated times, and estimated energy consumptions for each client $i \in \mathcal{C}_r$. The time $\mathtt{T}_i$ and energy $\mathtt{E}_i$ estimates are defined for each $x \in \widetilde{\mathtt{AC}}_{i,r}$. Let $\mathcal{L} = \{(\widetilde{L}^{(j)}_{i,\text{train}}, \widetilde{L}^{(j)}_{i,\text{test}})\}_{i \in \mathcal{C}_r,\, j=1,\ldots,q_i}$ denote the set of normalized historical loss values associated with each client $i \in \mathcal{C}_r$, including training and test losses.

Given a problem instance $(n, t, \tau, \mathcal{W}, \mathcal{A}, \mathcal{Y}, \mathcal{G}, \mathcal{E}, \mathcal{L})$, the goal is to find an optimal schedule that minimizes $F(\mathcal{X})$ according to the weights $\mathcal{W}$, as shown in Eq. (5.26).

$$
\mathcal{X}^* = \arg\min_{\mathcal{X}} F(\mathcal{X}) \tag{5.26}
$$

Here, $\mathcal{X}^*$ denotes the mathematical optimum of the formulated problem. MetaCS-FL

is metaheuristic-based and returns the best feasible solution found before its stopping condition. Unlike MEC and ECMTC, it does not claim a global optimality guarantee.

5.1.4 Illustrative Example

Consider an FL system with a server and three clients with heterogeneous resources. The server must allocate $t = 6$ tasks while meeting a deadline of $\tau = 15$ seconds. Each client i can process from 0 to 6 tasks, *i.e.*, $\widetilde{\mathtt{AC}}_{i,r} = \{0, 1, \ldots, 6\}$. For simplicity, all clients are assumed to be fully reliable, so $rs_i = 1$ and $\widetilde{\mathtt{AC}}_{i,r} = \mathtt{AC}_i$ for all clients. We also assume $\alpha = 0.5$ and $\beta = 0.5$, and that both the utility and diversity terms are computed based on the last $q = 2$ rounds. Accordingly, for each client i, the number of past participations considered is $q_i = \min\big(q, \sum_{\ell=1}^{r-1} \mathbb{I}(i \in \mathcal{S}_\ell)\big)$.

Table 12 summarizes the features for each feasible task assignment $x_i \in \widetilde{\mathtt{AC}}_{i,r}$. The values of u_i represent the contribution of client i to the learning process for a given workload x_i, increasing proportionally with the number of assigned tasks. The diversity component is not defined at the client level, but rather computed at the schedule level through D_r, based on the participation history of the selected clients. In particular, we consider the two rounds preceding round r, with client participation sets $\mathcal{S}_{r-2} = \{1,2\}$ and $\mathcal{S}_{r-1} = \{2,3\}$, which define the historical participation counts used in the computation of D_r. For each client i, the table reports the estimated processing time and energy consumption (δ_i, ϵ_i), the utility contribution u_i, and the local task distribution $\mathtt{Y}_i$.

Suppose the server uses a balanced strategy in which the workload is evenly assigned among the clients. In this case, each client processes two tasks, as shown in Fig. 11. Based on the estimates, the schedule $\mathcal{X} = [2, 2, 2]$ would have a makespan of $\mathrm{M}_r = \mathtt{T}_3(2) = 10.0$ seconds and a total energy consumption of $\Sigma_r = \mathtt{E}_1(2) + \mathtt{E}_2(2) + \mathtt{E}_3(2) = 13.32$ joules.

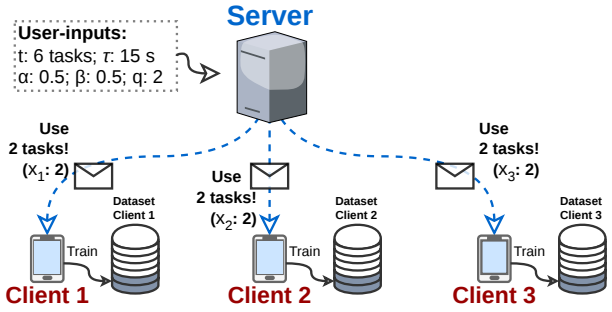

Figure 11: Server-client interaction concerning the distribution of the tasks.

Now, consider three sets of weights for the objective function: $\mathcal{W}_1 = \{0.2, 0.2, 0.2, 0.2, 0.2\}$ (equal priorities), $\mathcal{W}_2 = \{0.1, 0.9, 0, 0, 0\}$ (prioritizing energy consumption, followed by

Table 12: Client-specific features (illustrative example).

i	x_i	(δ_i, ϵ_i)	u_i	Υ_i
	0	$(0, 0)$	0.0	
	1	$(2.0, 3.5)$	0.13	
	2	$(4.0, 6.5)$	0.25	
1	3	$(6.0, 9.5)$	0.38	$[0, 1, 5]$
	4	$(8.0, 12.5)$	0.51	
	5	$(10.0, 15.5)$	0.63	
	6	$(12.0, 18.5)$	0.76	
	0	$(0, 0)$	0.0	
	1	$(3.0, 2.8)$	0.12	
	2	$(6.0, 5.2)$	0.24	
2	3	$(9.0, 7.6)$	0.35	$[2, 3, 1]$
	4	$(12.0, 10.0)$	0.47	
	5	$(15.0, 12.4)$	0.59	
	6	$(18.0, 14.8)$	0.71	
	0	$(0, 0)$	0.0	
	1	$(5.0, 0.88)$	0.13	
	2	$(10.0, 1.62)$	0.26	
3	3	$(15.0, 2.38)$	0.39	$[4, 2, 0]$
	4	$(20.0, 3.12)$	0.52	
	5	$(25.0, 3.88)$	0.65	
	6	$(30.0, 4.62)$	0.78	

makespan), and $\mathcal{W}_3 = \{0, 0, 0, 0.1, 0.9\}$ (prioritizing the utility score, followed by the class-distribution objective). A locally balanced distribution was applied for each selected client i, distributing the assigned tasks according to $\gamma_i \in \mathcal{Y}$. The optimal schedule $\mathcal{X}^*$ was chosen from 21 feasible schedules, obtained by enforcing the workload and deadline constraints over the 343 candidate schedules.

Table 13 summarizes the results. The obtained schedules reflect the trade-offs induced by the different weight settings.

Table 13: Optimal schedules (illustrative example).

Set of Weights	$\mathcal{X}^*$	$\mathbf{M}_r$	$\mathbf{\Sigma}_r$	$\mathbf{D}_r$	$\mathbf{K}_r$	$\mathbf{U}_r$
$\mathcal{W}_1$	[3, 2, 1]	6.00	15.58	0.98	0.55	0.75
$\mathcal{W}_2$	[0, 3, 3]	15.00	9.98	0.92	0.55	0.74
$\mathcal{W}_3$	[2, 1, 3]	15.00	11.68	0.98	0.67	0.76

Units: Σ_r in joules; M_r in seconds.

In particular, the balanced setting ($\mathcal{W}_1$) achieves the lowest makespan at the expense of higher energy consumption, while $\mathcal{W}_2$ minimizes energy by selecting fewer clients, resulting in a higher makespan and lower diversity. In contrast, $\mathcal{W}_3$ prioritizes utility and class distribution, leading to a more balanced allocation across clients. From this example, two observations emerge: (i) the objectives are inherently conflicting (*e.g.*, reducing makespan may increase energy consumption), and (ii) identifying optimal schedules be-

comes increasingly complex as the problem size grows.

For instance, with $n = 100$ clients, total workload $t = 20{,}000$, and $\widetilde{\mathtt{AC}}_{i,r} = \{0,1,\ldots,500\}$, the number of candidate schedules $\mathcal{X} = (x_1,\ldots,x_n)$ satisfying $\sum_{i=1}^{n} x_i = t$ and $x_i \in \widetilde{\mathtt{AC}}_{i,r}$ is the number of bounded weak compositions of t into n parts. By applying inclusion-exclusion to the upper-bound constraints (113), this number is

$$\sum_{k=0}^{\left\lfloor \frac{t}{501} \right\rfloor} (-1)^k \binom{n}{k} \binom{t - 501k + n - 1}{n - 1} \approx 6.49 \times 10^{263}$$

Enforcing $\mathrm{M}_r \leq \tau$ can only reduce this number; yet, brute-force exploration remains computationally infeasible.

5.2 MetaCS-FL

MetaCS-FL is a flexible framework for client selection in synchronous federated learning systems, designed to operate effectively under heterogeneous resource and data conditions. It promotes balanced and fair participation while jointly optimizing multiple objectives, including training time, energy consumption, and model quality. Within the federated learning loop, MetaCS-FL is invoked during the client selection steps, corresponding to ④ and ⑦ in Fig. 10. To efficiently explore the large combinatorial space of client-task assignments, MetaCS-FL combines structured scheduling with metaheuristic optimization. The framework follows an event-driven design, enabling dynamic adaptation to changes in system state and client behavior across rounds.

5.2.1 Workflow and Algorithm Overview

The workflow of MetaCS-FL is illustrated in Fig. 12 and summarized in Algorithm 7. At the start of each round, the framework updates the set of eligible clients, incorporates completed profiles from late-joining clients, and computes the reliability-aware task capacity set of each eligible client. It then determines whether a new client selection is required based on the current execution context, including changes in client availability and user-defined criteria derived from recent system behavior. If no new selection is needed, the framework retrieves the most recent selected clients and their corresponding task assignments from history and proceeds directly to training or testing.

If a new client selection is required, the framework determines whether a new initial solution must be generated or whether the most recent initial solution can be reused.

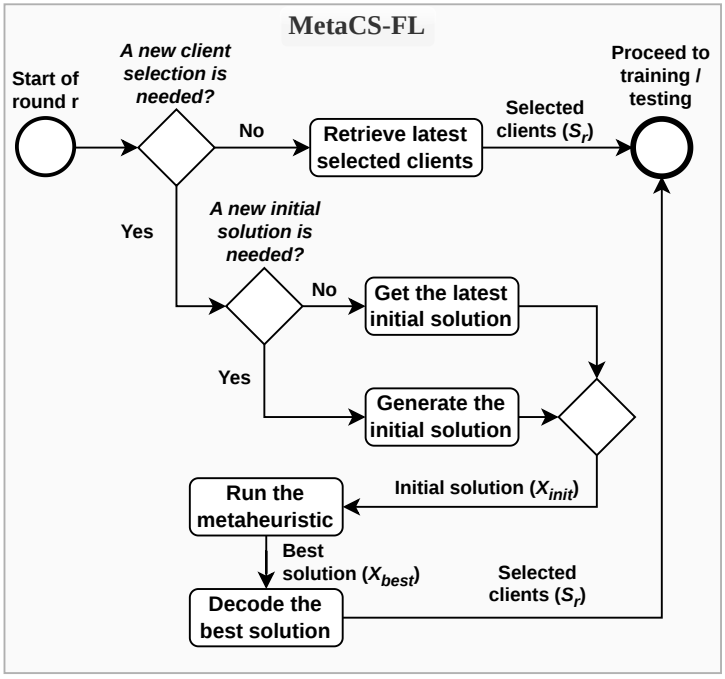

Figure 12: The workflow of MetaCS-FL.

Algorithm 7 MetaCS-FL

Initialize global model parameters $\mathbf{w}_0$
Profile the initial clients and store their profiles
for each round $r = 1,2,\ldots$ until stopping criterion is met **do**
    Update the eligible client set $\mathcal{C}_r$
    Incorporate completed profiles from late-joining clients into $\mathcal{C}_r$
    Compute rs_i and $\widetilde{\texttt{AC}}_{i,r}$ for each client $i \in \mathcal{C}_r$
    if a new client selection is needed **then**
        if a new initial solution is needed **then**
            Generate the initial solution $\mathcal{X}_{\text{init}}$
        else
            Get the latest initial solution $\mathcal{X}_{\text{init}}$
        Run the metaheuristic guided by $F(\mathcal{X})$ to obtain the best solution $\mathcal{X}_{\text{best}}$
        Decode $\mathcal{X}_{\text{best}}$ into selected clients $\mathcal{S}_r$ and task assignments x_i
    else
        Retrieve the latest selected clients $\mathcal{S}_r$ and task assignments x_i
    Server sends $\mathbf{w}_r$ and training instructions to each client $i \in \mathcal{S}_r$
    for each selected client $i \in \mathcal{S}_r$ in parallel **do**
        Initialize local model parameters $\mathbf{w}_r^i \leftarrow \mathbf{w}_r$
        Update $\mathbf{w}_r^i$ using x_i tasks sampled from $\mathcal{D}_i$ for E local epochs
        Send updated local model parameters $\mathbf{w}_{r+1}^i$ and metrics to the server
    Server aggregates the received local model parameters to obtain $\mathbf{w}_{r+1}$
    Server stores metrics and updates client histories and profiles

Reuse is allowed only when the set of candidate clients, *i.e.*, the available clients considered for selection, remains unchanged with respect to the previous selection; otherwise, a new initial solution is generated to reflect the updated availability conditions. When reuse is possible, it reduces overhead by avoiding the repeated generation of similar initial solutions. In this work, ECMTC (28) is used to generate the initial solution, as it minimizes energy consumption, followed by makespan, while satisfying a per-round deadline.

Once an initial solution for round r is available, the framework invokes the metaheuristic optimization procedure. Although MetaCS-FL supports different metaheuristics, in this work we employ *Large Neighborhood Search* (LNS) (72), a widely used method for complex routing and scheduling problems (114, 115, 116, 117). LNS balances intensification and diversification by iteratively destroying and repairing parts of a solution, enabling effective exploration of large combinatorial search spaces. Starting from the initial solution, the metaheuristic refines client-task assignments to optimize the objective function while respecting system constraints.

Figure 13 illustrates the use of LNS to schedule ten tasks ($t = 10$) across up to ten clients ($n = 10$). During this process, candidate solutions are evaluated according to the multi-objective formulation defined in Eq. (5.25). The search continues until a stopping condition is met, such as a time limit or a maximum number of iterations. The best solution found during the search is retained as the final solution.

Finally, the best solution is decoded to extract the set of selected clients $\mathcal{S}_r$ and their corresponding task assignments. These selected clients perform the training or testing phase of round r, after which the system state and relevant historical information (*e.g.*, client participation, performance metrics, and resource usage) are updated and stored. This updated history is used to guide client selection in subsequent rounds.

5.2.2 Computational Complexity

The computational complexity of MetaCS-FL depends on the initial solution method, the adopted metaheuristic, and the cost of evaluating candidate solutions. Given a problem instance, MetaCS-FL first obtains an initial solution and then refines it through a metaheuristic guided by the objective function $F(\mathcal{X})$ (Section 5.1.3.6).

Let C_{init} denote the cost of generating the initial solution, I_{MH} the number of metaheuristic iterations, and C_{eval} the cost of evaluating $F(\mathcal{X})$ for one candidate solution. In the worst case, when a new initial solution is generated and the metaheuristic is executed,

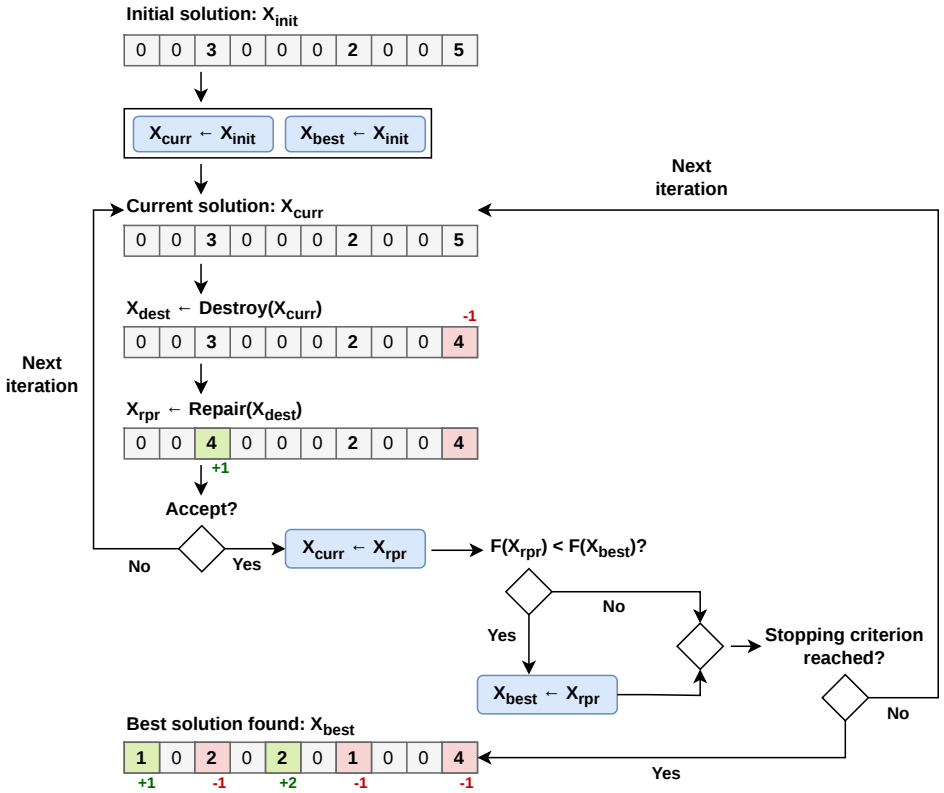

Figure 13: Illustrative LNS search for scheduling ten tasks across up to ten clients. Red and green squares indicate, respectively, decreases and increases in the number of tasks assigned to the ith client relative to the initial solution.

the client selection cost can be expressed as $\mathcal{O}\left(C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}\right)$.

In this work, C_{init} corresponds to the cost of ECMTC, which has a worst-case complexity of $\mathcal{O}(t^2 n)$, as discussed in Chapter 4. The term $I_{\text{MH}} \cdot C_{\text{eval}}$ captures the metaheuristic search cost, which depends on the considered method and its stopping condition, such as a time limit or a maximum number of iterations.

This worst-case cost applies only when a new client selection is required. Otherwise, MetaCS-FL retrieves the latest selected clients and task assignments from history in $\mathcal{O}(1)$ time, assuming direct access to the stored selection. When a new client selection is required but the previous initial solution can be reused, its generation cost is avoided, reducing the selection cost to $\mathcal{O}\left(I_{\text{MH}} \cdot C_{\text{eval}}\right)$. Thus, the framework avoids unnecessary computation by reusing prior selection information whenever possible. Nevertheless, these expressions describe one selection decision, not a cumulative time over all FL rounds. A runtime limit bounds the LNS refinement, but it does not bound the ECMTC initial-solution step, whose $\mathcal{O}(t^2 n)$ cost can dominate for large workloads. Increasing the number of local samples represented by one task reduces t and therefore this cost, at the expense of coarser workload control and a smaller assignment search space.

6 Experimental Evaluation

To evaluate the proposed framework, we conduct proof-of-concept experiments on realistic machine learning tasks using public datasets deployed across Grid'5000 computing nodes (107). All data and source code required for reproduction and further analysis are publicly available at https://github.com/alan-lira/metacs-fl.

The chapter is organized as follows. Section 6.1 describes the experimental setup, including datasets, models, emulated devices, network conditions, FL settings, benchmark algorithms, MetaCS-FL configuration, and performance metrics. Section 6.2 evaluates client selection under static availability, where all clients remain eligible for participation. This section also analyzes practical aspects of MetaCS-FL under static availability, including differentially private data-distribution reports, scalability with larger client pools and workloads, client-selection overhead, and parameter sensitivity. Section 6.3 evaluates dynamic availability scenarios, including late-joining and intermittent clients. Finally, Section 6.4 discusses the main findings across all experiments.

Throughout the evaluation, MetaCS-FL is compared against representative client selection algorithms: FedAvg (1), MEC (28), ECMTC (28), Oort (46), DivFL (15), ECSM (96), FedCAB (17), and RIFLES (22). These methods cover random selection, time- and energy-oriented scheduling, utility-driven selection, diversity-based selection, reputation-based selection, availability-budget selection, and availability-aware scheduling. This evaluation allows us to quantify the trade-offs among training time, energy consumption, convergence speed, fairness, privacy, scalability, and robustness under heterogeneous and dynamic FL conditions.

6.1 Experimental Setup

This section describes the experimental setup used to evaluate MetaCS-FL against representative client selection approaches. We first present the datasets and data partitioning schemes used to model IID and non-IID FL settings. We then describe the model archi-

tectures, emulated edge devices, network conditions, FL execution settings, benchmark algorithms, MetaCS-FL configuration, and performance metrics.

6.1.1 Datasets

Two widely adopted (79, 88, 9, 87, 97, 28, 99) image classification datasets were considered: CIFAR-10 (25) and Fashion-MNIST (118). CIFAR-10 contains 50,000 training images and 10,000 test images of 10 object categories, while Fashion-MNIST includes 60,000 training images and 10,000 test images representing 10 clothing categories. Compared to the original MNIST dataset (36), Fashion-MNIST exhibits greater complexity, making it a more challenging benchmark.

In addition, we consider the Emotion dataset (26), a text classification dataset comprising 417,000 samples across six emotion categories. As the original dataset does not provide predefined splits, we partition it into disjoint training and test sets using an 80/20 ratio, resulting in 333,600 training samples and 83,400 test samples. This dataset introduces a text-based modality, complementing the image-based benchmarks. We selected Emotion over Sentiment140 (119) due to its greater complexity and more reliable evaluation setting. Although Sentiment140 contains 1.6 million training samples, it is a binary classification task and provides only 359 manually annotated test samples.

6.1.2 Data Distributions

Two data partitioning schemes were used for each dataset: IID and non-IID. Under the IID setting, each client held samples from all classes in the dataset, with an equal number of samples per class (*i.e.*, 10 classes for CIFAR-10 and Fashion-MNIST, and 6 classes for Emotion). In contrast, the non-IID setting restricted each client to data from only two classes, with unequal sample counts, thereby introducing both class imbalance and distributional heterogeneity. The latter setting more closely reflects the conditions of real-world federated learning deployments (120, 14).

6.1.3 Models

To account for the resource limitations of mobile and edge devices, we adopted lightweight neural network architectures for all datasets.

For CIFAR-10, we employed the convolutional neural network (CNN) architecture

proposed by Nishio *et al.* (7). For Fashion-MNIST, we used a CNN based on the model of McMahan *et al.* (1), with minor modifications to improve computational efficiency without significantly compromising performance.

For Emotion, we adopted a bidirectional long short-term memory network with additive attention (BiLSTM-Attention), following the baseline described by Saravia *et al.* (26).

All models were trained using the Adam optimizer (35) with a learning rate of 0.001 and sparse categorical cross-entropy loss. Adam is a first-order stochastic optimizer that maintains exponentially decaying estimates of the first and second moments of the gradients, using them to adapt the update magnitude separately for each model parameter. In this work, Adam is applied independently by each selected client during local training and does not alter the server-side aggregation procedure. Local model weights were aggregated on the server using FedAvg (1), which computes a weighted average proportional to each client's data size. Dataset-specific training configurations were also applied. The CNN models used for CIFAR-10 and Fashion-MNIST were trained locally for five epochs with a batch size of 32, whereas the BiLSTM-Attention model used for Emotion was trained locally for four epochs with a batch size of 128.

6.1.4 Emulated Devices

Each client emulated a 4-core device drawn from a diverse set of platforms commonly used in embedded, IoT, and edge-AI applications, including the Coral Edge TPU, Atom x7-E3950 and x5-Z8350, Jetson TX1/TX2/Nano, Snapdragon 820E and 410E, and Raspberry Pi 3 and 4. Each client was assigned a fixed device type and geographic location for the duration of an execution. Device performance was emulated by profiling training and testing workloads on host machines to obtain baseline metrics, which were then analytically scaled using target hardware specifications (*e.g.*, FLOPs, memory bandwidth, CPU frequency, and device-specific power consumption). Although the assigned device type is unchanged, its execution time and energy consumption vary across rounds (see Table 39). This modeling-based approach follows the broader direction of FL simulation and benchmarking frameworks, such as FedML (48) and FedScale (50), which support controlled evaluation across simulated or heterogeneous execution environments.

Network conditions were simulated using an autoregressive (AR) model that varied bandwidth, latency, and packet loss across FL rounds, reflecting typical 4G LTE dynamics. Clients were also assigned to 12 geographic locations, introducing distance-based latency and jitter. Client locations remain fixed (see Figure 32), while communication conditions

vary across rounds, producing round-dependent bandwidth and latency.

The experiments do not assume background applications or explicitly emulate contention for client CPU, memory, or network interfaces. Consequently, the round-to-round variation described above represents the modeled device and network behavior rather than interference from co-located application workloads.

6.1.5 FL Settings

Experiments were conducted on systems with up to 50 and 100 clients, depending on the client availability conditions. Each server-scheduled task is an atomic workload unit measured in local samples and represents a single image sample for CIFAR-10 and Fashion-MNIST, and a batch of 10 text samples for the Emotion dataset. The set of task capacities per client ranged from zero to the full local dataset size in steps of 50. In each round, 40% of training samples and all test samples were scheduled to the selected clients.

Equal task size refers only to the amount of local data processed and does not imply statistically identical or IID content. For class-aware selection, the server determines both the total assignment and its estimated class-specific composition from the exact or DP-perturbed client histogram. Each selected client then draws the requested records from its own local partition, as raw samples and data ownership remain local.

Clients are lightly profiled upon joining the system using 200 training and 200 testing tasks, enabling the server to estimate their performance under different task loads for client selection. This profiling occurs synchronously before the first FL round for initially available clients and asynchronously for late-joining clients upon connection. Thereafter, successfully completed phases supply the current observations used to refresh these estimates, including communication measurements. Clients that have not participated recently may still be evaluated using an older record or the initial fallback profile.

To ensure privacy, the server queries client data distributions through the Differential Privacy (DP) approach (27). Specifically, clients report noisy class histograms using a privacy parameter $\epsilon = 0.1$, which corresponds to a strong privacy regime. This setting is applied consistently across all evaluated methods, including MetaCS-FL and benchmark algorithms, even when such methods do not explicitly leverage data distribution information, ensuring a consistent and fair comparison.

Following (7), the target test accuracies were set to 0.75/0.45 for CIFAR-10 and 0.85/0.7 for Fashion-MNIST under IID and non-IID settings, respectively. For Emotion,

the targets were set to 0.9 and 0.8, respectively, based on typical results reported in the literature (26, 121). The FL execution was terminated upon reaching the target accuracy or after 100 FL rounds, whichever occurred first. The per-round deadline τ was set to 1800 seconds for CIFAR-10, 300 seconds for Fashion-MNIST, and 1500 seconds for Emotion. For every evaluated method, a selected client is considered to have completed its assigned processing only if it returns a valid result by τ. Results received after the deadline are discarded and excluded from the corresponding training aggregation or testing record, while the associated unfinished workload is recorded as failed. For MetaCS-FL, such a deadline miss additionally incurs the completion penalty λ_{compl} when reliability is updated, which may restrict the client's admissible task capacities in later selections.

6.1.6 Benchmark Algorithms

To evaluate the effectiveness of *MetaCS-FL* in complex and heterogeneous environments, we compare it against representative client selection algorithms, as summarized below:

- **FedAvg** (1) (Baseline): randomly selects a fraction of available clients each round and aggregates their local updates via simple averaging.
- **MEC** (28): an optimal scheduler that minimizes training time, followed by energy consumption.
- **ECMTC** (28): an optimal scheduler that minimizes energy consumption, then training time, while also satisfying a per-round deadline.
- **Oort** (46): selects clients by balancing estimated training utility and execution efficiency, aiming to accelerate convergence under system and data heterogeneity.
- **DivFL** (15): selects clients by maximizing diversity via a facility-location objective defined over client gradient updates.
- **ECSM** (96): selects clients through an ensemble strategy, combining historical accuracy, client reputation, and random sampling.
- **FedCAB** (17): ranks available clients using communication budgets, participation frequency, statistical heterogeneity, and late-joining behavior.
- **RIFLES** (22): formulates client selection as an availability-aware scheduling problem, using heartbeat-based forecasts and historical participation to guide selection.

For the static client availability evaluation (Section 6.2), we compare *MetaCS-FL* against FedAvg, MEC, ECMTC, Oort, DivFL, and ECSM. FedAvg serves as the baseline, while the remaining methods cover optimal time- and energy-oriented scheduling and representative utility-, diversity-, and ensemble-based client selection strategies.

For the dynamic client availability evaluation (Section 6.3), we compare *MetaCS-FL* against FedAvg, Oort, DivFL, ECSM, FedCAB, and RIFLES. FedAvg again serves as the baseline; Oort, DivFL, and ECSM are availability-agnostic comparison methods; and FedCAB and RIFLES explicitly account for client availability dynamics.

To ensure a fair comparison within our experimental setting, we adapt the benchmark algorithms only where necessary to align them with the server-defined workload model and the available information at client-selection time.

Accordingly, FedAvg, Oort, DivFL, ECSM, FedCAB, and RIFLES are extended to control workload allocation according to the server-defined workload per round. In all cases, client selection is followed by the same workload-assignment mechanism used by MetaCS-FL, which assigns feasible task quantities to the selected clients.

For ECSM, we adopt the weighting scheme of 20% accuracy, 70% reputation, and 10% random selection, as it was the best-performing configuration in the original work (96).

For FedCAB, we preserve its rank-based client weighting mechanism, which accounts for data heterogeneity, client participation, and late-joining behavior, following the example in the original paper (17). Since the original method computes KL divergence between local model updates and the global model, we replace this term with class-distribution KL divergence to enable pre-training client selection in our evaluation setting.

For RIFLES, we consider the Greedy Heuristic variant, denoted as RIFLES-GH, since it is reported to outperform the Least Recently Used (LRU) variant (22). We preserve its eligibility-based, availability-aware scheduling principle, but replace the original heartbeat-driven CNN-LSTM availability predictor with an estimator based on the availability records collected by the server as FL training progresses.

6.1.7 MetaCS-FL Settings

A new client selection is triggered if: (i) the current round follows initial profiling; (ii) the set of candidate clients changes; or (iii) history-based criteria are met. During training, the criteria are a client-diversity score below 0.85, a relative drop exceeding 20% over 3

rounds, or increases in makespan or energy exceeding 10% over 2 rounds. During testing, the criterion is a decrease in accuracy exceeding 10% over 2 rounds.

To reduce overhead, MetaCS-FL reuses the most recent initial solution for up to 10 consecutive rounds when the candidate client set remains unchanged. Otherwise, a new initial solution is generated to reflect updated client availability. The objective function $F(\mathcal{X})$ uses weights $\mathcal{W} = \{0.05, 0.05, 0.3, 0.2, 0.4\}$, with $\alpha = 0.35$ (utility, prioritizing test loss) and $\beta = 0.7$ (class distribution, prioritizing coverage). Since the ECMTC initial solution already prioritizes energy and makespan, the main configuration assigns greater weight to participation diversity, class-distribution quality, and utility so that the refinement can explore learning-oriented alternatives. However, these values define a controlled baseline and are not claimed to be universally optimal. Utility and diversity are computed using the last $q = 3$ rounds.

For the dynamic client availability evaluation, the reliability score uses a non-availability penalty $\lambda_{\text{avail}} = 0.3$, a non-completion penalty $\lambda_{\text{compl}} = 0.7$, and a memory weight $\mu = 0.6$. These settings penalize incomplete processing more strongly than temporary unavailability, while retaining 60% of the previous score to smooth short-term fluctuations. These values serve as penalty settings, not probabilities inferred from a production disconnection trace, and are used to test the scheduling response under controlled availability and completion outcomes. See Appendix B.5 for more details on the availability modeling.

For LNS, a 10-second execution time limit is set per call. The *destroy phase* selects 30% of clients and removes 15%–75% of their assigned tasks, while the *repair phase* reconstructs feasible solutions. Repaired solutions are accepted only if client diversity improves by at least 0.5% and makespan and energy increase by at most 25% relative to the initial solution; the best solution is updated whenever the objective value improves.

6.1.8 Performance Metrics

We evaluated five performance metrics: total training time ($\mathrm{M}_{\text{total}}$), total training energy (Σ_{total}), total client selection time (T_{cs}), fairness of client selection (S_{fair}), and the number of rounds required to reach a target test accuracy x (RoA@x). $\mathrm{M}_{\text{total}}$ and Σ_{total} denote the cumulative time and energy consumed to reach the target test accuracy, while T_{cs} measures the total time spent performing client selection across all rounds and is, therefore, a cumulative execution-level quantity, not the latency of a single selection decision. S_{fair}, computed using Jain's Fairness Index (54), quantifies how evenly clients are selected, with values closer to 1 indicating greater fairness. Specifically, if z_i is the number of rounds

in which client i is selected, then $S_{\text{fair}} = (\sum_i z_i)^2/(n \sum_i z_i^2)$. This external evaluation metric is distinct from the entropy-based participation-diversity term D_r optimized by MetaCS-FL. Test accuracy is computed as the weighted mean of correctly classified local test samples over the total number of test samples. Each experiment was repeated three times, and we report the mean of each metric.

Importantly, the evaluation does not report an aggregate utilization rate for all available edge resources. Scheduled workload, failed workload, energy, fairness, and server-side overhead characterize related aspects, but they do not support a claim that an approach maximizes total edge-resource utilization.

6.2 Evaluation Under Static Client Availability

In this set of experiments, we evaluate the behavior of the FL system under a *static client availability* setting. Specifically, we consider systems composed of 50 and 100 clients that are all available from the first communication round and remain continuously connected throughout the entire training process. Before entering the FL loop, all clients undergo a lightweight profiling phase consisting of 200 training and 200 testing tasks, enabling the server to estimate their performance under varying task loads during client selection.

Although all clients are consistently available, only a subset participates in each round according to the client selection strategy. This setup represents a controlled environment with no client arrivals, departures, or intermittent connectivity, allowing us to isolate and analyze the performance of each method over a fixed and fully reliable pool of eligible clients. The results in this section, therefore, serve as a baseline under stable conditions.

6.2.1 Client Participation and Workload Allocation

To answer **RQ1**, this subsection examines how MEC, ECMTC, and MetaCS-FL affect client participation and workload allocation across communication rounds compared with FedAvg and state-of-the-art client selection strategies.

Tables 14 and 15 summarize how the client selection algorithms distributed client participation and training workload. Table 14 reports the mean number of clients selected per round, while Table 15 shows the mean number of processed samples assigned to each selected client. For reporting comparability, task assignments are converted to sample counts: one task corresponds to one image sample for CIFAR-10 and Fashion-MNIST

and ten text samples for Emotion. Since each round processes a fixed 40% of the global training data, the scheduling behavior of each algorithm directly impacts data diversity, update variance, and ultimately, model convergence.

Table 14: Mean number of selected clients per round.

System Size	CS Approach	CIFAR-10 IID	CIFAR-10 Non-IID	Fashion-MNIST IID	Fashion-MNIST Non-IID	Emotion IID	Emotion Non-IID
50	FedAvg	25.0	26.0	25.0	26.0	25.0	25.0
	MEC	44.05	44.11	42.33	38.73	41.31	43.71
	ECMTC	20.0	20.35	20.0	20.67	21.0	17.29
	Oort	20.77	20.54	24.76	21.03	21.66	19.26
	DivFL	20.0	21.0	20.0	21.0	21.0	23.0
	ECSM	20.0	20.83	20.0	19.93	21.0	17.51
	MetaCS-FL	25.0	25.18	24.33	25.21	25.93	22.08
100	FedAvg	50.0	49.0	54.0	50.0	50.0	50.0
	MEC	76.42	71.31	71.08	68.31	99.03	99.5
	ECMTC	40.0	39.67	40.0	39.33	41.0	31.42
	Oort	40.59	40.96	42.55	41.21	41.42	39.59
	DivFL	40.0	41.0	40.0	41.67	41.0	42.0
	ECSM	40.0	42.58	40.0	40.03	41.0	36.22
	MetaCS-FL	47.14	47.36	44.6	47.6	49.18	43.54

Table 15: Mean scheduled workload (processed samples) per client per round.

System Size	CS Approach	CIFAR-10 IID	CIFAR-10 Non-IID	Fashion-MNIST IID	Fashion-MNIST Non-IID	Emotion IID	Emotion Non-IID
50	FedAvg	800.0	769.23	960.0	923.08	5337.6	5337.6
	MEC	454.72	454.15	575.16	635.58	3365.3	3087.76
	ECMTC	1000.0	983.17	1200.0	1161.9	6354.29	7722.67
	Oort	984.6	985.96	1085.71	1158.29	6185.9	6948.42
	DivFL	1000.0	952.38	1200.0	1142.86	6354.29	5809.06
	ECSM	1000.0	961.56	1200.0	1207.23	6354.29	7696.36
	MetaCS-FL	800.0	794.86	991.65	953.17	5145.99	6179.99
100	FedAvg	400.0	408.16	444.44	480.0	2668.8	2668.8
	MEC	262.5	283.32	343.54	352.75	1347.7	1341.19
	ECMTC	500.0	504.27	600.0	610.26	3254.63	4247.72
	Oort	496.79	494.61	577.54	589.76	3225.15	3379.67
	DivFL	500.0	487.8	600.0	576.07	3254.63	3178.34
	ECSM	500.0	469.94	600.0	600.24	3254.63	3703.52
	MetaCS-FL	425.8	423.43	542.96	505.58	2719.02	3087.84

FedAvg selects a fixed random fraction of clients and assigns an equal number of samples to each selected client. As a result, its mean workload per client follows directly from the number of selected clients in each setting. This produces stable and predictable task allocation, but it does not account for system or data heterogeneity.

MEC, which optimizes training time and energy consumption, selects the largest number of clients per round in most scenarios. As a result, it assigns the fewest samples per selected client, especially in the 100-client Emotion setting, where it selects nearly all clients and reduces the workload to approximately 1347 and 1341 tasks under IID and non-IID data, respectively. MEC favors shorter local execution times by distributing the workload across a broader client pool.

In contrast, ECMTC prioritizes energy minimization under per-round deadlines by selecting fewer clients and assigning larger workloads to each participant. This pattern appears across all datasets, with per-client workloads reaching 1000 samples for CIFAR-10 IID, 1200 for Fashion-MNIST IID, and over 6000 tasks for Emotion IID in the 50-client setting. Under Emotion non-IID, ECMTC assigns the largest workloads, reaching approximately 7723 tasks with 50 clients and 4248 with 100 clients.

Oort selects clients by balancing estimated training utility and execution efficiency, leading to relatively high per-client workloads. In most scenarios, its workloads are close to those of ECMTC, DivFL, and ECSM, especially with 50 clients. This suggests that Oort concentrates training on a smaller client subset, which may improve utility-driven convergence but also increase workload imbalance.

DivFL also assigns large workloads per selected client, reflecting its tendency to select a relatively small number of clients while enforcing diversity-oriented selection. Its workload levels are close to those of ECMTC in several IID settings, including CIFAR-10, Fashion-MNIST, and Emotion. However, in non-IID settings, its workload varies depending on the selected diverse subset, particularly for Emotion, where it assigns approximately 5809 tasks per client with 50 clients and 3178 tasks with 100 clients.

ECSM follows a similar pattern of concentrated workload allocation, generally selecting fewer clients and assigning larger numbers of tasks per participant. This is particularly evident in the Emotion non-IID setting, where ECSM assigns approximately 7696 tasks per client with 50 clients and 3704 tasks with 100 clients. These results reflect the effect of its ensemble strategy, which combines accuracy, reputation, and random sampling, leading to workload concentration on selected clients.

MetaCS-FL maintains an intermediate workload allocation profile. Compared with MEC, it assigns more samples per selected client because it does not simply maximize participation. However, compared with ECMTC, Oort, DivFL, and ECSM, it generally avoids excessive workload concentration. For example, in the 100-client setting, MetaCS-FL assigns approximately 426 and 423 samples per client for CIFAR-10 IID and non-IID,

543 and 506 for Fashion-MNIST IID and non-IID, and 2719 and 3088 for Emotion IID and non-IID. Overall, MetaCS-FL combines moderate client participation with controlled per-client task assignments across datasets and data distributions.

6.2.2 Impact on Performance Metrics

To answer **RQ2** and **RQ3**, this subsection evaluates how MetaCS-FL affects system- and model-level performance under IID and non-IID data distributions, and compares it with MEC, ECMTC, and existing methods in terms of test accuracy, convergence speed, total training time, and total energy consumption across image and text classification domains.

For brevity, the performance of the algorithms was evaluated on the 100-client system. Shaded areas in the figures indicate the standard deviation over three independent runs.

6.2.2.1 CIFAR-10 IID

Table 16 reports the performance of the client selection algorithms on CIFAR-10 IID, while Figure 14 shows the progression of their test accuracy over the FL rounds.

Table 16: Performances for CIFAR-10 IID (100-client system).

CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.75
FedAvg (Baseline)	53,145.17	5,114.37	3.24	0.98	50.33 ± 1.25
MEC	21,324.51 (−59.88%)	7,596.38 (+48.53%)	1897.1 (+58,539.4%)	0.79 (−19.54%)	—
ECMTC	65,144.84 (+22.58%)	6,383.31 (+24.81%)	2848.79 (+87,955.98%)	0.4 (−59.22%)	—
Oort	83,479.45 (+57.08%)	7,228.18 (+41.33%)	12.17 (+276.11%)	0.56 (−43.07%)	88.0 ± 1.41
DivFL	102,860.55 (+93.55%)	7,905.11 (+54.57%)	2026.15 (+62,528.38%)	0.4 (−59.22%)	—
ECSM	50,608.38 (−4.77%)	3,909.37 (−23.56%)	4.81 (+48.67%)	0.64 (−34.5%)	58.33 ± 5.44
MetaCS-FL	15,077.9 (−71.63%)	1,912.49 (−62.61%)	317.96 (+9728.19%)	0.69 (−29.28%)	37.0 ± 0.82

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

MetaCS-FL achieved the lowest total training time (15,077.9 s, −71.63% vs. FedAvg) and energy consumption (1,912.49 kJ, −62.61%), balancing efficiency and fairness (0.69). It also converged the fastest, reaching 75% test accuracy in just 37 rounds.

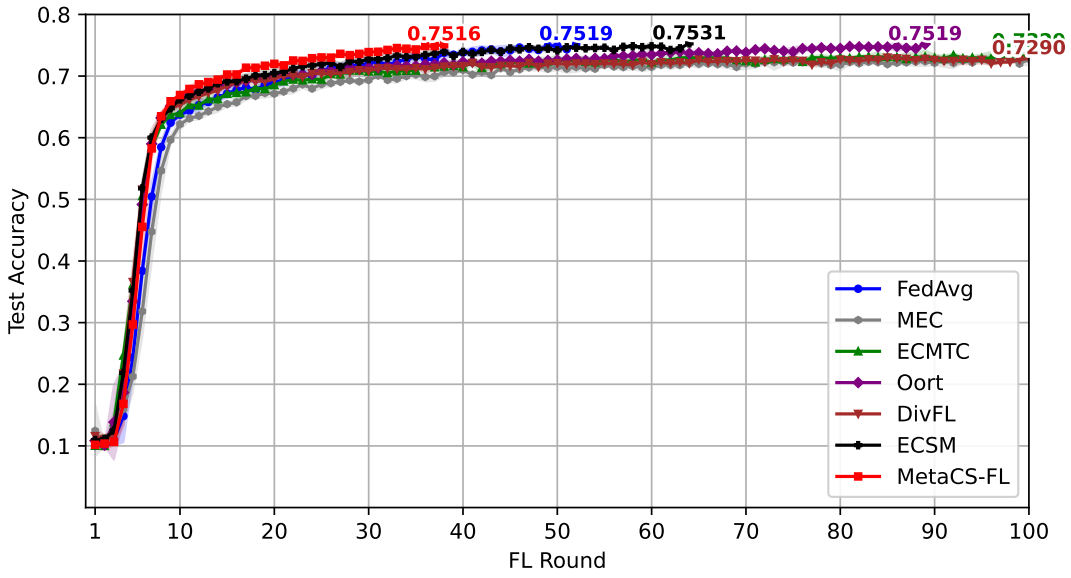

Figure 14: Test accuracy progression for the 100-client system on CIFAR-10 IID, with 20,000 training samples processed per round.

FedAvg achieved the highest fairness score (0.98) while maintaining a moderate training time and convergence speed (≈50 rounds) and minimal selection overhead (3.24 s).

MEC substantially reduced training time (−59.88%), but incurred high energy consumption (+48.53%) and extreme selection time. Fairness remained relatively high (0.79) due to near-full client participation. ECMTC showed low fairness (0.4) due to repeated selection of a small, non-diverse client subset. Oort had moderate selection overhead (12.17 s) and converged more slowly than FedAvg (88 rounds).

DivFL incurred the highest training time (+93.55%) and energy consumption (+54.57%), while maintaining low fairness (0.4) and failing to converge. In contrast, ECSM reduced both training time and energy consumption (−4.77% and −23.56%, respectively), achieved moderate fairness (0.64), and reached the target accuracy within 58 rounds.

Notably, MEC, ECMTC, and DivFL failed to reach the target accuracy within the measured rounds, suggesting that prioritizing client selection for energy, time, or diversity by choosing clients that best represent the overall population can hinder global model convergence, even when the data are IID.

6.2.2.2 CIFAR-10 non-IID

Table 17 summarizes the performance of the client selection algorithms on CIFAR-10 non-IID, while Figure 15 shows the progression of their test accuracy over the FL rounds.

Table 17: Performances for CIFAR-10 non-IID (100-client system).

CS Approach	$\mathbf{M_{total}}$	$\mathbf{\Sigma_{total}}$	$\mathbf{T_{cs}}$	$\mathbf{S_{fair}}$	**RoA@0.45**
FedAvg (Baseline)	98,158.41	9,201.13	160.23	0.99	92.0 ± 8.0
MEC	20,195.06 (−79.43%)	6,963.07 (−24.32%)	2295.67 (+1332.72%)	0.74 (−25.39%)	—
ECMTC	66,321.67 (−32.43%)	5,887.11 (−36.02%)	2921.86 (+1723.52%)	0.4 (−59.88%)	—
Oort	50,337.49 (−48.72%)	5,235.03 (−43.1%)	294.73 (+83.94%)	0.52 (−47.74%)	57.7 ± 1.7
DivFL	53,676.83 (−45.32%)	4,108.17 (−55.35%)	1482.18 (+825.02%)	0.41 (−58.53%)	61.33 ± 9.46
ECSM	72,341.44 (−26.3%)	5,482.77 (−40.41%)	320.1 (+99.77%)	0.59 (−40.08%)	86.67 ± 10.27
MetaCS-FL	10,947.75 (−88.85%)	1,430.92 (−84.45%)	294.39 (+83.73%)	0.65 (−34.7%)	29.67 ± 4.03

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

MetaCS-FL secured the lowest total training time, 10,947.75 s (a reduction of 88.85% compared to FedAvg), and the lowest energy consumption, 1,430.92 kJ (a reduction of 84.45%), while maintaining a strong balance between efficiency and fairness (0.65). It also converged the fastest, requiring only 29 rounds to reach 45% test accuracy.

FedAvg achieved the highest fairness score (0.99), but incurred a long training time (98,158.41 s). While benefiting from the lowest selection overhead (160.23 s), it required 92 rounds to reach the target accuracy.

MEC reduced training time (−79.43%), but had higher energy consumption (6,963.07 kJ, −24.32%) and a very high selection overhead (2295.67 s), with moderate fairness (0.74) due to near-full client participation. ECMTC saved energy (−36.02%), but suffered low fairness (0.40) and extreme selection time (2921.86 s). Neither reached the target accuracy, likely due to poor adaptation to non-IID data.

Oort reduced training time and energy consumption by 48.72% and 43.1%, respectively, but introduced a moderate selection overhead of 294.73 s and required more rounds (≈ 58) to converge compared to MetaCS-FL.

DivFL reduced both training time (−45.32%) and energy consumption (−55.35%), but maintained low fairness (0.41) and only reached the target accuracy after 61 rounds. ECSM showed smaller efficiency gains (−26.3% time, −40.41% energy), with moderate fairness (0.59), but required significantly more rounds (86) to converge.

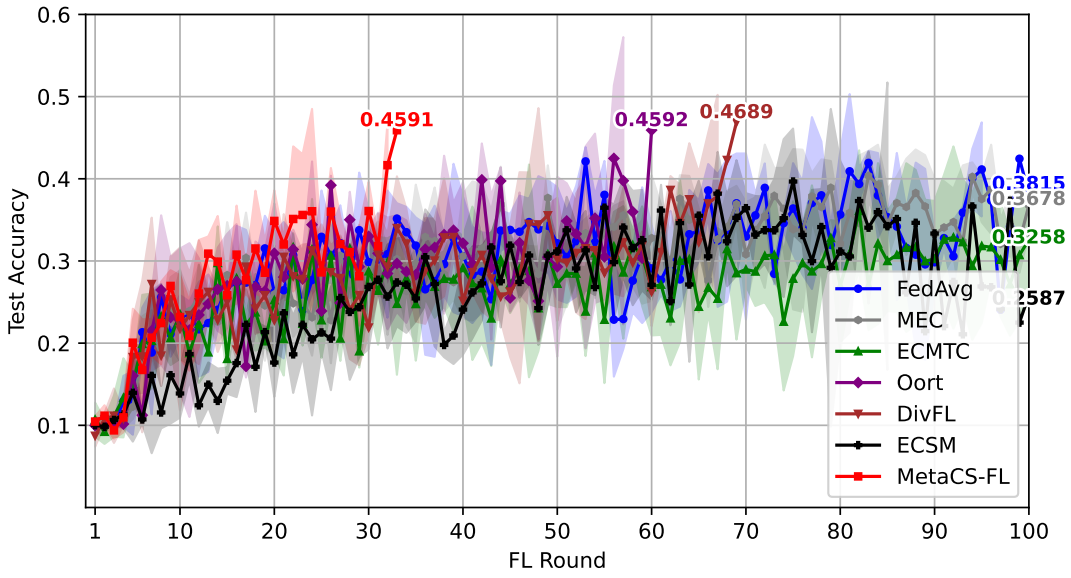

Figure 15: Test accuracy progression for the 100-client system on CIFAR-10 non-IID, with 20,000 training samples processed per round.

6.2.2.3 Fashion-MNIST IID

Table 18 details the performance of the client selection algorithms on Fashion-MNIST IID, while Figure 16 shows the progression of their test accuracy over the FL rounds.

Table 18: Performances for Fashion-MNIST IID (100-client system).

CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.85
FedAvg (Baseline)	3771.77	400.21	1.39	0.96	18.33 ± 0.47
MEC	1164.91 (−69.12%)	317.37 (−20.7%)	390.61 (+27,949.88%)	0.8 (−16.45%)	15.33 ± 0.47
ECMTC	1584.45 (−57.99%)	167.60 (−58.12%)	459.77 (+32,916.78%)	0.4 (−58.33%)	12.33 ± 0.47
Oort	1869.93 (−50.42%)	202.82 (−49.32%)	1.73 (+24.13%)	0.47 (−51.09%)	11.0 ± 0
DivFL	2536.9 (−32.74%)	205.02 (−48.77%)	98.46 (+6970.59%)	0.4 (−58.33%)	12.33 ± 0.47
ECSM	2395.87 (−36.48%)	187.87 (−53.06%)	1.35 (−3.39%)	0.6 (−37.4%)	13.67 ± 0.47
MetaCS-FL	964.25 (−74.44%)	112.08 (−72.0%)	57.42 (+4023.53%)	0.58 (−39.33%)	10.0 ± 0.0

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

MetaCS-FL achieved the lowest total training time (964.25 s, −74.44% vs. FedAvg) and significant energy savings (112.08 kJ, −72.0%), while maintaining moderate fairness (0.58). It also converged the fastest, reaching 85% test accuracy in just 10 rounds.

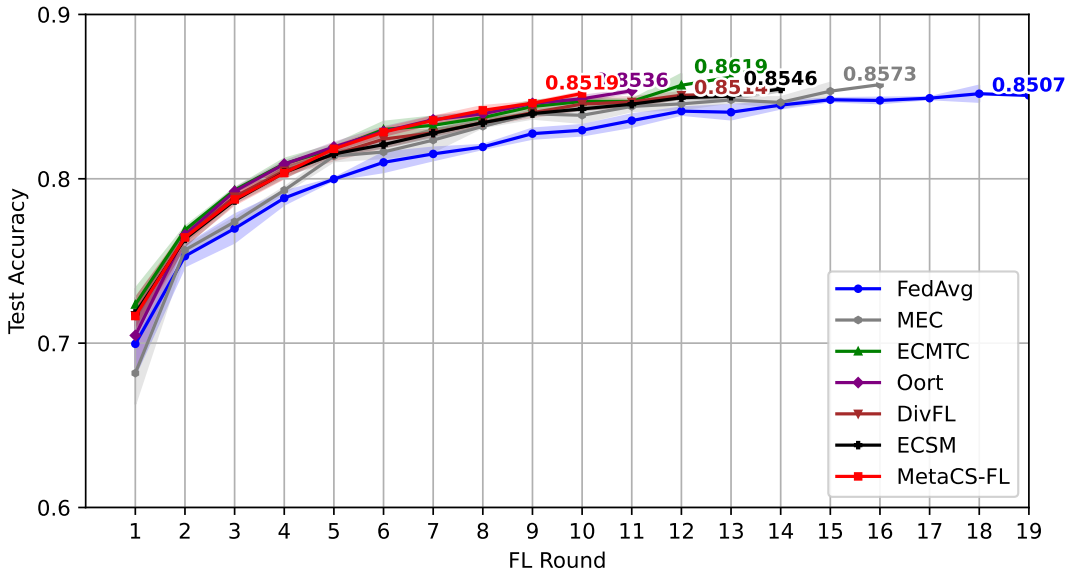

Figure 16: Test accuracy progression for the 100-client system on Fashion-MNIST IID, with 24,000 training samples processed per round.

FedAvg had the highest fairness (0.96), but needed longer training time (3771.77 s) and more rounds to converge (≈ 18), with minimal selection overhead (1.39 s).

MEC substantially reduced training time (−69.12%) and moderately lowered energy consumption (−20.7%), but incurred a very high selection overhead (390.61 s) while maintaining relatively high fairness (0.8). ECMTC achieved significant energy savings (−58.12%) but suffered from very low fairness (0.4), higher selection overhead (459.77 s), and slower effective convergence (≈ 12 rounds).

Oort reduced both training time and energy consumption (−50.42% and −49.32%) with negligible selection overhead (1.73 s). It converged in 11 rounds, matching MetaCS-FL, but exhibited low fairness (0.47).

DivFL reduced both training time (−32.74%) and energy consumption (−48.77%), but maintained low fairness (0.4) and required about 12 rounds to reach the target accuracy. ECSM achieved greater energy savings (−53.06%), with moderate fairness (0.6) and minimal selection overhead, requiring around 13 rounds to converge.

6.2.2.4 Fashion-MNIST non-IID

Table 19 provides the performance of the client selection algorithms on Fashion-MNIST non-IID, while Figure 17 shows the progression of their test accuracy over the FL rounds.

MetaCS-FL achieved the fastest training (1998.92 s, −85.78% vs. FedAvg) and the

Table 19: Performances for Fashion-MNIST non-IID (100-client system).

CS Approach	$\mathbf{M_{total}}$	$\mathbf{\Sigma_{total}}$	$\mathbf{T_{cs}}$	$\mathbf{S_{fair}}$	**RoA@0.7**
FedAvg (Baseline)	14,055.85	1,410.47	155.17	0.98	65.0 ± 0.82
MEC	5194.14 (−63.05%)	1,688.32 (+19.7%)	2619.02 (+1587.88%)	0.72 (−27.07%)	87.0 ± 9.2
ECMTC	8343.36 (−40.64%)	857.23 (−39.22%)	2532.05 (+1531.83%)	0.39 (−60.03%)	64.67 ± 10.5
Oort	9655.5 (−31.31%)	818.92 (−41.94%)	267.64 (+72.48%)	0.47 (−52.46%)	41.33 ± 4.19
DivFL	13,968.86 (−0.62%)	1,048.61 (−25.66%)	974.7 (+528.16%)	0.42 (−57.66%)	73.33 ± 8.18
ECSM	16,079.94 (+14.4%)	1,427.07 (+1.18%)	440.73 (+184.04%)	0.51 (−47.98%)	—
MetaCS-FL	1998.92 (−85.78%)	239.98 (−82.99%)	279.25 (+79.97%)	0.68 (−30.56%)	20.67 ± 0.94

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

lowest energy consumption (239.98 kJ, −82.99%), while maintaining reasonable fairness (0.68). It reached 70% test accuracy in just 20 rounds, demonstrating robustness under imbalanced data distributions.

FedAvg achieved the highest fairness (0.98) and the lowest selection overhead (155.17 s), but required longer training time (14,055.85 s) and more rounds to converge (65).

MEC reduced training time (−63.05%), but increased energy consumption (+19.7%) and incurred higher selection overhead (2619.02 s), while preserving moderate fairness (0.72). ECMTC saved 39.22% energy, but incurred the lowest fairness (0.39), high selection overhead (2532.05 s), and slower convergence than FedAvg (≈ 65 rounds).

Oort reduced training time and energy (−31.31% and −41.94%) with moderate selection overhead (267.64 s). However, it converged slower than MetaCS-FL (≈ 41 rounds) and had moderate fairness (0.47).

DivFL showed limited training time reduction (−0.62%) but reduced energy consumption (−25.66%), while maintaining low fairness (0.42) and converging in about 73 rounds. In contrast, ECSM increased both training time (+14.4%) and energy consumption (+1.18%), with moderate fairness (0.51), and failed to reach the target accuracy.

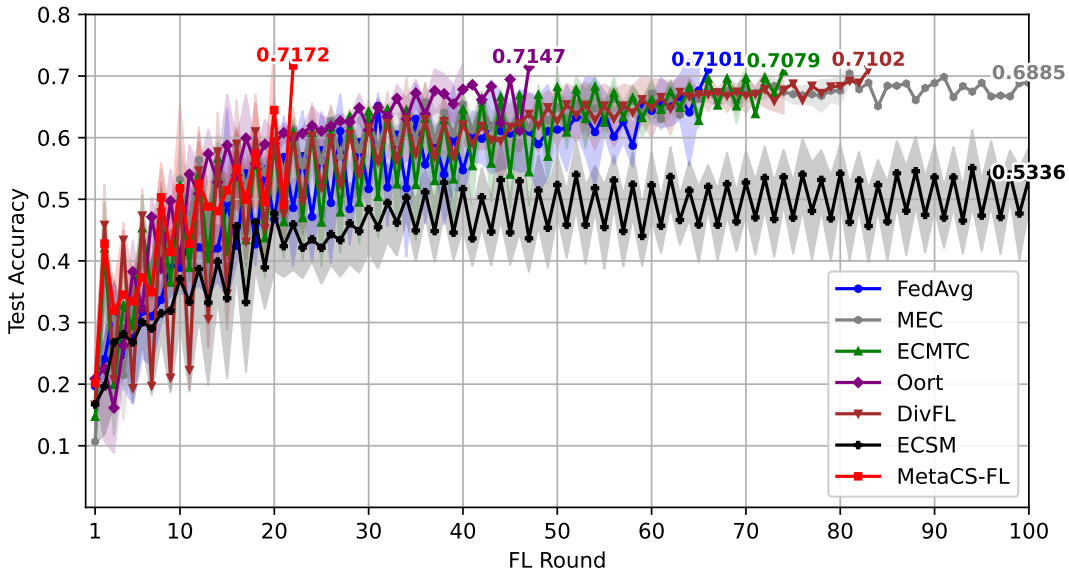

Figure 17: Test accuracy progression for the 100-client system on Fashion-MNIST non-IID, with 24,000 training samples processed per round.

6.2.2.5 Emotion IID

Table 20 details the performance of the client selection algorithms on Emotion IID, while Figure 18 shows the progression of their test accuracy over the FL rounds.

Table 20: Performances for Emotion IID (100-client system).

CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.9
FedAvg (Baseline)	13,889.33	1,251.88	58.26	0.92	13.33 ± 0.47
MEC	11,049.08 (−20.45%)	1,832.35 (+46.37%)	1,650.43 (+2732.97%)	0.99 (+8.02%)	21.0 ± 0.82
ECMTC	7,628.27 (−45.08%)	710.63 (−43.24%)	1,385.74 (+2278.63%)	0.41 (−55.53%)	12.67 ± 0.47
Oort	14,125.24 (+1.7%)	1,175.07 (−6.14%)	163.13 (+180.01%)	0.43 (−52.9%)	12.0 ± 0.0
DivFL	12,909.64 (−7.05%)	1,057.06 (−15.56%)	496.35 (+751.98%)	0.41 (−55.53%)	10.33 ± 0.47
ECSM	11,682.53 (−15.89%)	860.14 (−31.29%)	93.95 (+61.27%)	0.64 (−30.57%)	9.67 ± 0.47
MetaCS-FL	6,602.25 (−52.47%)	1,013.83 (−19.02%)	201.49 (+245.85%)	0.51 (−44.51%)	11.0 ± 0.0

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

MetaCS-FL achieved the lowest total training time, requiring 6,602.25 s to reach the 90% accuracy target, which corresponds to a 52.47% reduction compared with FedAvg.

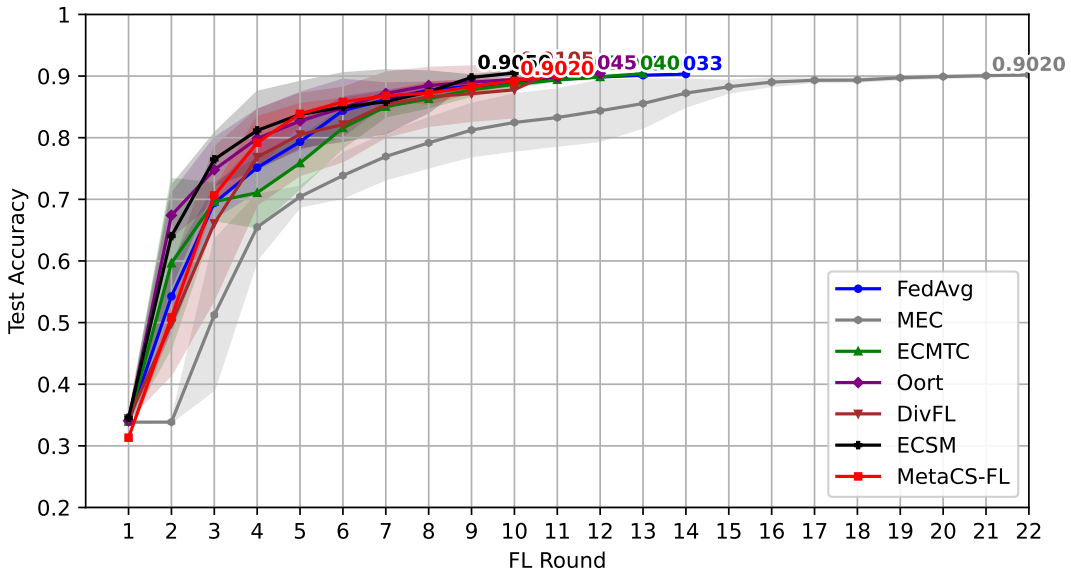

Figure 18: Test accuracy progression for the 100-client system on Emotion IID, with 133,440 training samples processed per round.

It also reduced energy consumption by 19.02%, with a total of 1,013.83 kJ, and required 11 rounds for convergence. With a moderate fairness score of 0.51, these results indicate that MetaCS-FL effectively prioritizes efficient and representative clients, substantially reducing training time without excessive workload concentration.

FedAvg maintained high fairness (0.92), with moderate training time (13,889.33 s) and the lowest client selection overhead (58.26 s). However, it required 13 rounds to reach the target accuracy, showing that uniform random participation does not necessarily provide the most efficient convergence.

MEC reduced total training time by 20.45%, requiring 11,049.08 s, but increased energy consumption by 46.37% and incurred the highest client selection overhead (1,650.43 s). Although it achieved the highest fairness score (0.99), it required 21 rounds for convergence, indicating that selecting nearly all clients improves fairness but can lead to less effective updates and higher system cost.

ECMTC achieved strong reductions in both training time and energy consumption, decreasing them by 45.08% and 43.24%, respectively. However, this came at the cost of very low fairness (0.41) and high selection overhead (1,385.74 s). Its convergence speed of 13 rounds was only slightly better than FedAvg, suggesting that its energy-oriented scheduling improves resource efficiency but offers limited gains in convergence.

Oort slightly increased training time by 1.7% while reducing energy consumption by 6.14%. It required 12 rounds to reach the target accuracy and achieved low fairness (0.43).

These results suggest that prioritizing high-utility clients did not provide significant gains in this IID Emotion setting.

DivFL reduced the number of rounds to 10, outperforming FedAvg in convergence speed. However, its total training time remained relatively high at 12,909.64 s, with only a 7.05% reduction compared with FedAvg. It also showed low fairness (0.41) and a substantial selection overhead of 496.35 s, indicating that diversity-oriented selection improved convergence in rounds but did not translate into proportional time savings.

ECSM provided the fastest convergence, requiring only 10 rounds to reach the target accuracy. It also reduced training time by 15.89% and energy use by 31.29%, while keeping selection overhead low at 93.95 s. Compared with DivFL and ECMTC, ECSM offered a better trade-off among convergence speed, energy efficiency, overhead, and fairness, although its fairness remained lower than FedAvg and MEC.

6.2.2.6 Emotion non-IID

Table 21 details the performance of the client selection algorithms on Emotion non-IID, while Figure 19 shows the progression of their test accuracy over the FL rounds.

Table 21: Performances for Emotion non-IID (100-client system).

CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.8
FedAvg (Baseline)	29,934.81	2,187.34	176.71	0.97	34.0 ± 2.83
MEC	17,006.53 (−43.19%)	2,628.32 (+20.16%)	3,450.43 (+1852.57%)	0.99 (+2.33%)	43.67 ± 3.09
ECMTC	24,552.72 (−17.98%)	2,463.63 (+12.63%)	6,129.34 (+3368.55%)	0.31 (−67.67%)	56.33 ± 2.62
Oort	26,565.69 (−11.25%)	2,014.46 (−7.9%)	438.69 (+148.25%)	0.48 (−50.51%)	33.67 ± 1.25
DivFL	45,766.62 (+52.89%)	2,222.5 (+1.61%)	1,803.25 (+920.45%)	0.42 (−56.88%)	34.33 ± 2.62
ECSM	83,682.43 (+179.55%)	5,945.37 (+171.81%)	843.0 (+377.05%)	0.47 (−51.41%)	93.33 ± 3.86
MetaCS-FL	15,087.27 (−49.6%)	1,756.58 (−19.69%)	539.0 (+205.02%)	0.49 (−49.8%)	23.67 ± 0.47

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

MetaCS-FL achieved the lowest total training time, requiring 15,087.27 s to reach the 80% accuracy target, which corresponds to a 49.6% reduction compared with FedAvg. It also achieved the lowest energy consumption, with 1,756.58 kJ, representing a 19.69%

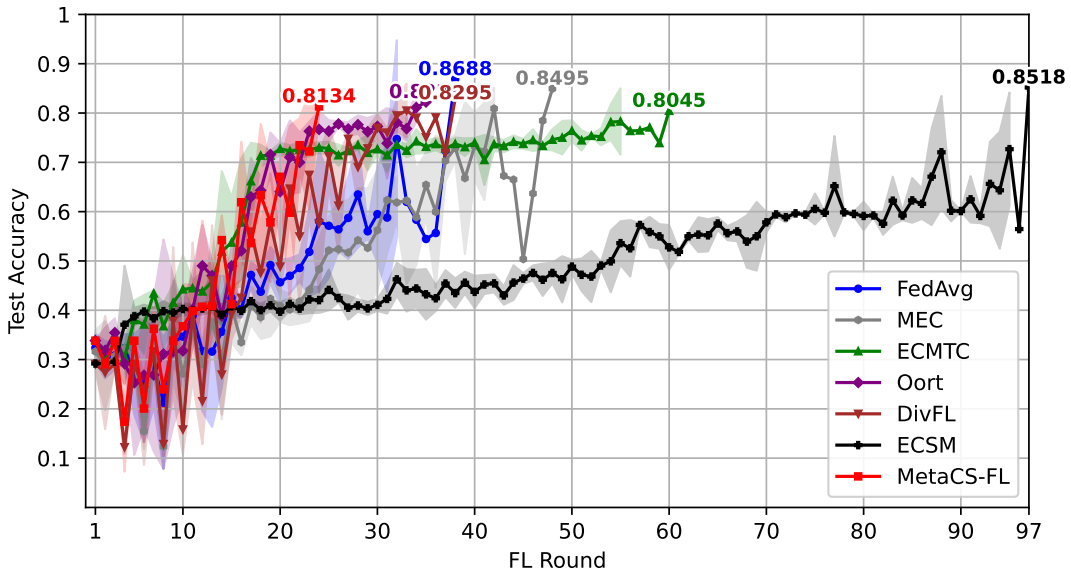

Figure 19: Test accuracy progression for the 100-client system on Emotion non-IID, with 133,440 training samples processed per round.

reduction. In addition, MetaCS-FL required only 24 rounds for convergence, outperforming all other approaches in this metric. With a moderate fairness score of 0.49, these results show that MetaCS-FL provides the best overall trade-off under Emotion non-IID by substantially reducing training time, energy consumption, and convergence rounds.

FedAvg maintained high fairness (0.97), with moderate selection overhead (176.71 s). However, it required 29,934.81 s and 34 rounds to reach the target accuracy, reflecting the cost of uniform random participation under non-IID data.

MEC reduced total training time by 43.19%, requiring 17,006.53 s, but increased energy consumption by 20.16% and incurred a very high selection overhead of 3,450.43 s. Although it achieved the highest fairness score (0.99), it required 44 rounds for convergence, suggesting that selecting nearly all clients improves fairness but can limit convergence efficiency under non-IID data.

ECMTC reduced training time by 17.98%, requiring 24,552.72 s, but increased energy consumption by 12.63% and produced the lowest fairness score (0.31). It also incurred the highest selection overhead, with 6,129.34 s, and required 56 rounds for convergence. These results suggest that its energy-oriented scheduling becomes less effective under Emotion non-IID due to concentrated client participation and biased local updates.

Oort reduced training time by 11.25% and energy consumption by 7.9%, requiring 26,565.69 s and 2,014.46 kJ. However, its fairness score remained moderate (0.48), and it required 34 rounds for convergence, close to FedAvg. This indicates that prioritizing

high-utility clients provides limited benefit under this heterogeneous setting.

DivFL required 34 rounds for convergence, similar to FedAvg, but increased total training time by 52.89% and energy consumption by 1.61%. It also incurred substantial selection overhead (1,803.25 s) and low fairness (0.42). These results indicate that diversity-oriented selection did not translate into efficiency gains for Emotion non-IID.

ECSM performed poorly in this setting, requiring 83,682.43 s and 93 rounds to reach the target accuracy. It also increased energy consumption by 171.81% and incurred 843.0 s of selection overhead. Although its fairness score (0.47) was comparable to Oort and MetaCS-FL, its convergence behavior indicates that its ensemble-based selection was not effective under this non-IID Emotion scenario.

6.2.2.7 Discussion on performance metrics

Across all datasets, MetaCS-FL consistently demonstrated the strongest overall balance between efficiency, convergence speed, and fairness. It achieved the lowest total training time (M_{total}) and energy consumption (Σ_{total}) in most IID and non-IID scenarios, including CIFAR-10, Fashion-MNIST, and Emotion. In the few cases where another method was competitive in one metric, MetaCS-FL still provided the best combined trade-off by jointly reducing training time, energy consumption, and rounds to reach the target accuracy. Its fairness scores remained moderate across the experiments ($S_{fair} \approx 0.49$–0.69), indicating that the efficiency gains were achieved without extreme workload concentration. Moreover, MetaCS-FL reached the target accuracy in the fewest rounds in almost all settings, showing robustness to both IID and non-IID data distributions.

FedAvg consistently achieved the highest fairness, with S_{fair} values close to or above 0.92 across all scenarios. This behavior is expected because FedAvg relies on uniform client participation, distributing the training workload more evenly over time. However, this fairness came at the cost of longer total training time and higher energy consumption, particularly in the non-IID settings. Although FedAvg generally incurred the lowest or near-lowest selection overhead, its lack of client prioritization limited its efficiency and convergence speed compared with adaptive selection strategies.

MEC reduced total training time in several scenarios by selecting a large fraction of clients and favoring broad participation. This often resulted in high fairness, especially on the Emotion dataset, where MEC achieved the highest fairness in both IID and non-IID settings. However, MEC also incurred extremely high client selection overhead and

frequently increased energy consumption, particularly on CIFAR-10 IID, Fashion-MNIST non-IID, and both Emotion settings. In addition, MEC failed to reach the target accuracy in some CIFAR-10 scenarios and converged slowly in others, suggesting that selecting many clients does not necessarily produce efficient or high-quality global updates.

ECMTC generally reduced energy consumption in some settings, particularly on Fashion-MNIST and CIFAR-10 non-IID, but this came with very low fairness and substantial client selection overhead. Its repeated selection of a small subset of clients led to workload concentration and likely introduced biased model updates, especially under heterogeneous data distributions. On Emotion non-IID, ECMTC even increased energy consumption while requiring many more rounds than MetaCS-FL, indicating that its energy-oriented scheduling was not robust across datasets.

Oort provided moderate reductions in training time and energy consumption in several scenarios and maintained lower selection overhead than MEC, ECMTC, and DivFL. However, its fairness remained low to moderate because it prioritized high-utility clients more frequently. While Oort often converged faster than FedAvg, it was generally slower than MetaCS-FL and did not consistently deliver the same system-wide efficiency. On Emotion, Oort showed limited benefits: in the IID case, it slightly increased total training time, and in the non-IID case, its convergence speed was close to FedAvg.

DivFL, despite explicitly maximizing diversity, did not consistently translate this diversity into improved overall performance. In some non-IID scenarios, such as CIFAR-10 non-IID, it reduced both training time and energy consumption and reached the target accuracy. However, fairness remained low, and its selection overhead was high. On Emotion non-IID, DivFL increased total training time and energy consumption while converging at a similar rate to FedAvg, indicating that diversity-oriented selection alone is insufficient when system efficiency, client cost, and learning utility are not jointly optimized.

ECSM showed mixed behavior across datasets. In some IID settings, such as Fashion-MNIST IID and Emotion IID, it provided a reasonable trade-off by reducing energy consumption, maintaining moderate fairness, and keeping selection overhead relatively controlled. However, its convergence and efficiency were inconsistent under non-IID distributions. It failed to reach the target accuracy on Fashion-MNIST non-IID and performed particularly poorly on Emotion non-IID, where it substantially increased training time, energy consumption, and convergence rounds. These results suggest that its ensemble-based selection strategy can be effective in simpler or more homogeneous settings, but may become unstable under stronger data heterogeneity.

6.2.3 Impact of Differential Privacy on MetaCS-FL

To answer **RQ4**, this subsection examines how DP-based disclosure of client data-distribution information affects representation, convergence, fairness, and system performance compared with the MetaCS-FL variant without DP-based protection.

For brevity, the impact of employing Differential Privacy (DP) is evaluated in the 100-client CIFAR-10 non-IID setting, which is the most challenging scenario.

6.2.3.1 Comparison of server-side knowledge of data classes

Figures 20, 21, and 22 compare the server's knowledge under non-private and DP settings for the overall data distribution, the most-affected client, and the least-affected client, respectively. Client sensitivity to DP noise is quantified using the ℓ_1 divergence between the true and reported class-count vectors, defined as the sum of absolute per-class differences; larger values indicate greater distortion due to DP.

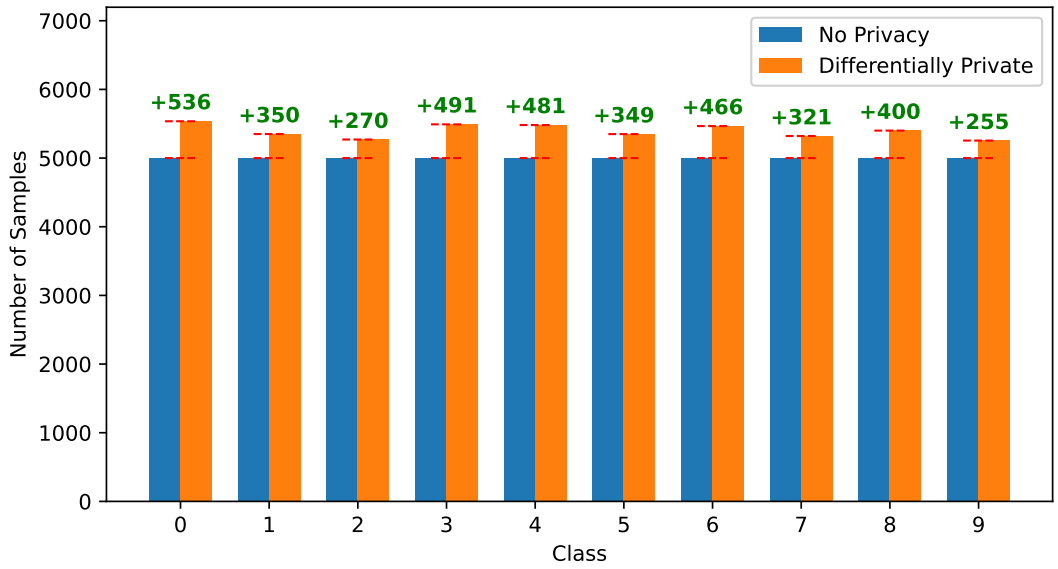

Figure 20: Overall class distributions under non-private and DP cases.

At the overall level (Figure 20), the non-private setting is perfectly balanced, with 5000 samples per class (totaling 50,000). In contrast, the DP case reports 53,919 samples, a 7.84% increase, unevenly distributed across classes. The per-class deviations are moderate, ranging from +255 for class 9 to +536 for class 0, showing that DP noise perturbs counts in a relatively balanced way across all classes.

The effect of DP-induced noise is more pronounced at the client level. For the most

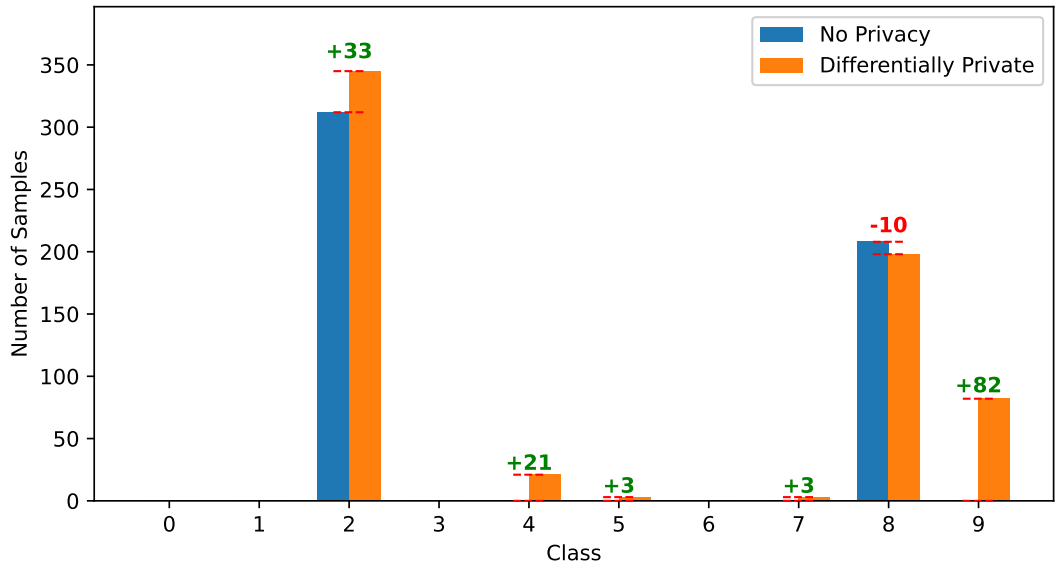

Figure 21: Class distribution of the client most affected by DP-induced noise.

affected client (Figure 21), the non-private data contains 520 samples primarily from classes 2 and 8. Under DP, the reported total increases to 652, with spurious counts added to previously absent classes and small reductions in the true majority classes, yielding an ℓ_1 divergence of 152 and a DP impact ratio of 0.29. In contrast, the least affected client (Figure 22) shows a smaller shift: its 431 samples, mostly in classes 4 and 6, are only slightly perturbed by DP, resulting in an ℓ_1 divergence of 8 and a DP impact ratio of 0.02.

On average, the clients experience a DP impact ratio of 0.12, indicating that approximately 12% of each client's data is affected by noise. These observed distortions directly affect server-side decisions, as clients are often selected or weighted according to their reported class distributions to promote diversity, mitigate class imbalance, or accelerate convergence, as explicitly optimized in MetaCS-FL. Nevertheless, such protective mechanisms remain essential to uphold data privacy.

6.2.3.2 Comparison of performance metrics

Table 22 presents the performance of MetaCS-FL on CIFAR-10 non-IID under non-private and DP cases, while Figure 23 shows the progression of test accuracy over the FL rounds.

The results indicate that DP introduces mixed overheads in MetaCS-FL compared to the no-privacy baseline. Total training time slightly decreases by 0.65%, while energy consumption increases significantly by 23.87%, and client selection time increases by 12.36%. Moreover, the number of rounds required to reach the target accuracy increases

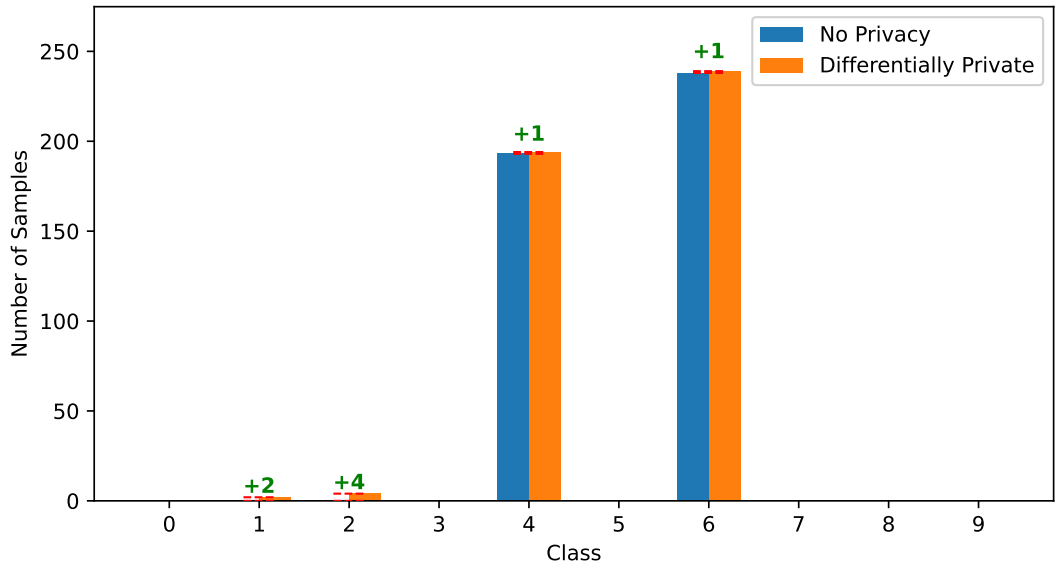

Figure 22: Class distribution of the least-affected client by DP-induced noise.

Table 22: Performances of MetaCS-FL under non-private and DP cases.

CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.45
MetaCS-FL No Privacy (Baseline)	11,019.08	1,155.19	262.01	0.59	26.33 ± 4.5
MetaCS-FL	10,947.75 (−0.65%)	1,430.92 (+23.87%)	294.39 (+12.36%)	0.65 (+8.88%)	29.67 ± 4.03

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

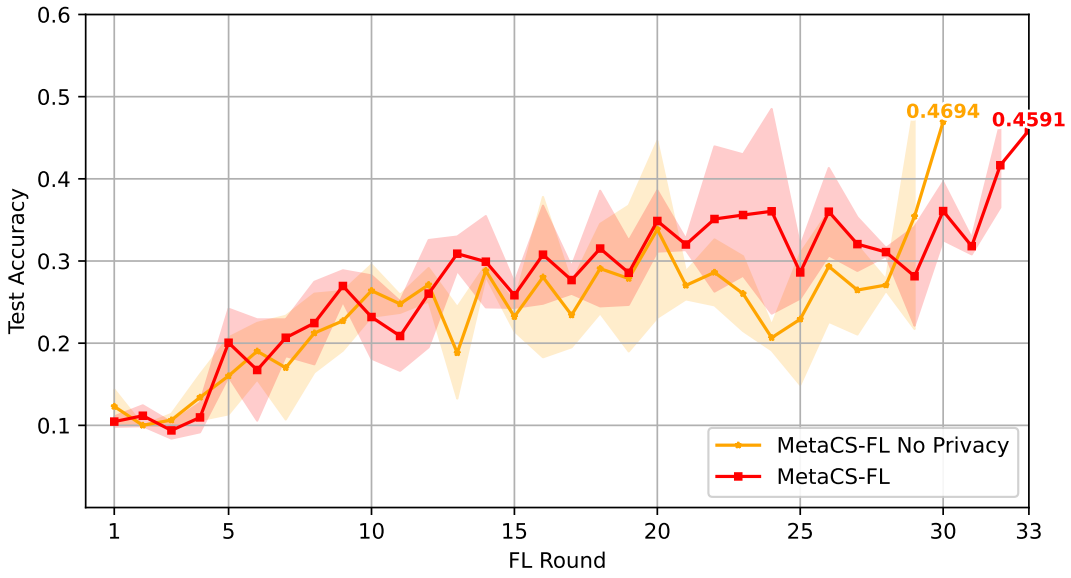

Figure 23: Test accuracy progression for the 100-client system on CIFAR-10 non-IID, with 20,000 training samples processed per round.

from about 26 to 29, corresponding to an increase of less than 12%, reflecting slower convergence in terms of communication rounds. Still, fairness improves by 8.88%, indicating more balanced participation. Despite these effects, performance remains close to the no-privacy case, where exact client class distributions are available to the server, highlighting MetaCS-FL's effectiveness in balancing privacy preservation and system efficiency.

6.2.4 Scalability of Client Selection Algorithms

To answer **RQ5**, this subsection evaluates how MetaCS-FL scales in terms of client selection time as the number of tasks and available clients increases, compared with existing client selection methods.

We conducted simulation-based experiments to evaluate the scalability of MetaCS-FL against the benchmark algorithms.

While our approach relies on user-defined thresholds to trigger client selection (Section 5.2.1), the scalability experiments were designed to be threshold-agnostic, ensuring evaluation independent of any specific threshold. Accordingly, MetaCS-FL was executed in three independent runs with fixed probabilities of 0.15 (low), 0.25 (moderate), and 0.5 (stress test) to trigger client selection each round.

For workload allocation purposes, all experiments considered a representation of a dataset containing 50,000 training samples and 10,000 test samples across 10 classes. All simulations were executed for 100 rounds, meaning that the total client selection overhead (in seconds) is the sum over all rounds. Consequently, the figures and tables in this subsection report cumulative campaign-level overhead, not a 20–30-minute delay for one synchronous selection decision. Since the approaches invoke selection a different number of times and MetaCS-FL may reuse a prior solution, even dividing these totals by 100 yields a per-round average that includes rounds without a new optimization and is not a per-call latency. The performance metrics of each client were generated by varying the collected results from the previous executions (Section 6.2.2).

6.2.4.1 Fixed number of clients, varying number of tasks

The scalability was assessed by keeping the number of clients fixed and gradually increasing the workload from 100 to 40,000 tasks. The evaluation considered a range of client pools: 10, 50, 100, 500, and 1000, covering all corresponding client-task combinations. For brevity, we focus on detailed results for the 500- and 1000-client cases.

For the 500-client case, Figure 24 shows the total client selection overhead (in seconds) across all workload sizes, while Table 23 reports a representative subset: small (100), medium (1000), large (10,000), and very large (30,000 and 40,000).

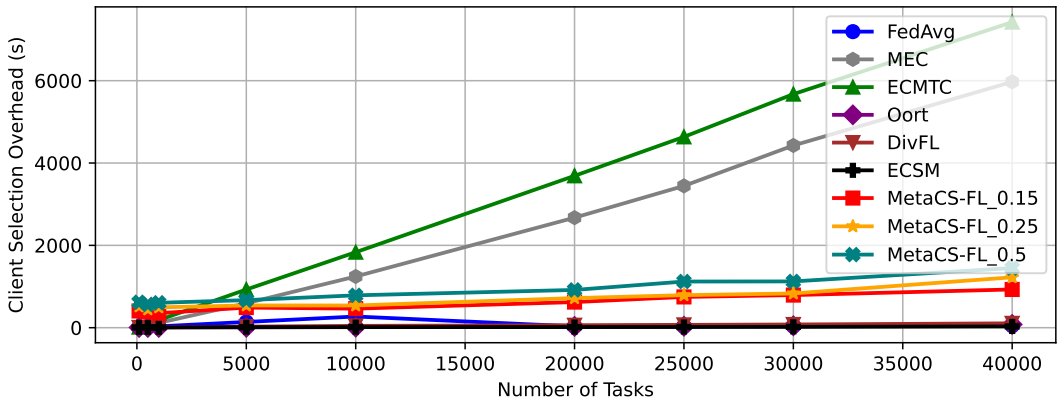

Figure 24: Cumulative selection overhead across all workload sizes over 100 simulated rounds (500 fixed clients).

Table 23: Cumulative selection overhead for the representative subset of workload sizes (500 fixed clients). Lowest cumulative overheads for each workload size are highlighted in bold.

CS Approach	Number of Tasks 100	1000	10,000	30,000	40,000
FedAvg	8.8	32.8	271.8	36.7	47.3
MEC	13.1	116.6	1244.1	4426.5	5972.1
ECMTC	14.5	184.4	1833.8	5673.4	7422.1
Oort	**6.1**	**7.3**	14.1	45.9	79.0
DivFL	6.9	10.8	44.3	78.0	106.1
ECSM	7.2	7.9	**13.9**	**19.9**	**31.1**
MetaCS-FL_0.15	406.8	351.7	462.5	792.2	928.9
MetaCS-FL_0.25	517.3	494.3	539.8	831.8	1224.1
MetaCS-FL_0.5	617.8	604.5	786.2	1124.2	1448.2

Oort, DivFL, and ECSM consistently achieved the lowest overhead for small and medium workloads (*e.g.*, 6.1–10.8 s at 100–1000 tasks), followed by FedAvg (8.8–32.8 s). MEC and ECMTC scaled poorly, reaching several thousand seconds for 30,000–40,000 tasks. MetaCS-FL variants showed higher overhead for small workloads (406.8–617.8 s), but remained more efficient than MEC and ECMTC at larger scales (792.2–1448.2 s). For large workloads (10,000–40,000 tasks), ECSM consistently achieved the lowest overhead.

For the 1000-client case, Figure 25 shows the total client selection overhead (in seconds) across all workload sizes, while Table 24 reports a representative subset: small

(100), medium (1000), large (10,000), and very large (30,000 and 40,000).

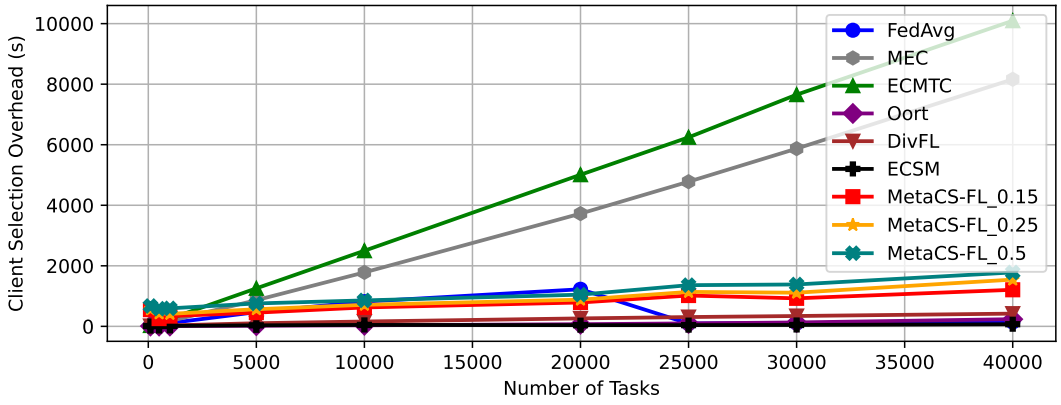

Figure 25: Cumulative selection overhead across all workload sizes over 100 simulated rounds (1000 fixed clients).

Table 24: Cumulative selection overhead for the representative subset of workload sizes (1000 fixed clients). Lowest cumulative overheads for each workload size are highlighted in bold.

CS Approach	Number of Tasks 100	1000	10,000	30,000	40,000
FedAvg	19.5	108.7	815.9	109.9	143.0
MEC	22.0	174.8	1782.6	5871.2	8161.6
ECMTC	27.0	251.6	2498.2	7653.2	10095.4
Oort	10.7	**11.7**	**35.6**	129.1	237.1
DivFL	12.7	30.1	160.0	341.8	420.5
ECSM	**10.0**	16.4	47.6	**45.5**	**70.9**
MetaCS-FL_0.15	551.8	338.5	624.2	925.7	1208.1
MetaCS-FL_0.25	561.6	413.0	712.0	1111.5	1544.8
MetaCS-FL_0.5	673.1	595.0	857.5	1383.9	1781.2

Similarly to the 500-client case, Oort and ECSM achieved the lowest overhead depending on the workload scale, with ECSM performing best for small and large workloads (*e.g.*, 10.0 s at 100 tasks and 70.9 s at 40,000 tasks), and Oort excelling at medium scales (11.7 s at 1,000 tasks and 35.6 s at 10,000 tasks). FedAvg exhibited higher and non-monotonic overhead, while DivFL remained generally above Oort and ECSM, particularly for larger workloads. MEC and ECMTC had moderate overhead for small tasks, but rose sharply for larger workloads, reaching several thousand seconds at 30,000–40,000 tasks. MetaCS-FL variants started with higher overheads (551.8–673.1 s at 100 tasks), but scaled efficiently for larger workloads, achieving approximately 925.7–1781.2 s at 30,000–40,000 tasks.

6.2.4.2 Fixed number of tasks, varying number of clients

The scalability was assessed by fixing the number of tasks and gradually increasing the number of candidate clients from 10 to 1000. The evaluation considered a range of workload sizes: 100, 500, 1000, 5000, 10,000, 20,000, 25,000, 30,000, and 40,000 tasks. For brevity, we focus on results for the larger workloads of 30,000 and 40,000 tasks.

Figure 26 and Table 25 present the total client selection overhead (in seconds) across different client pool sizes for the 30,000-task case.

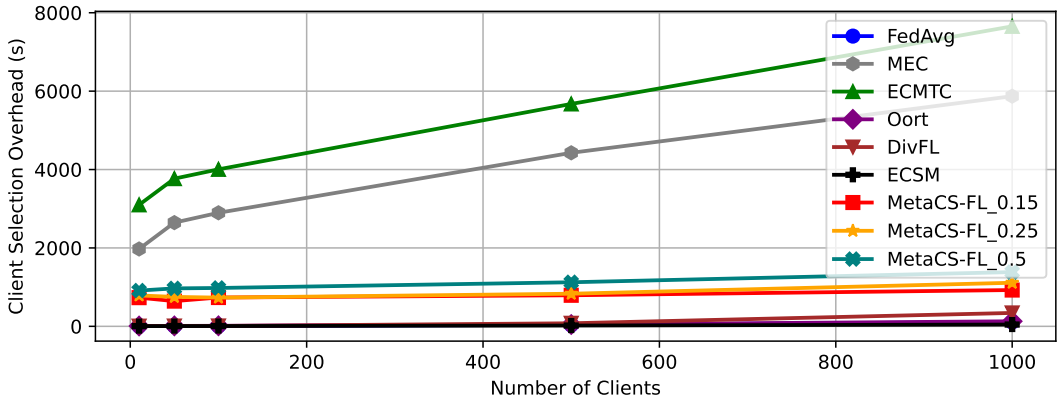

Figure 26: Cumulative selection overhead across all client pools over 100 simulated rounds (30,000 fixed workload).

Table 25: Cumulative selection overhead for 30,000 tasks with varying numbers of clients. Lowest cumulative overheads for each client count are highlighted in bold.

CS Approach	Number of Clients 10	50	100	500	1000
FedAvg	6.7	7.8	9.3	36.7	109.9
MEC	1973.2	2645.8	2896.4	4426.5	5871.2
ECMTC	3099.2	3767.7	4004.9	5673.4	7653.2
Oort	8.7	11.1	13.7	46.0	129.1
DivFL	5.2	7.7	9.6	78.0	341.8
ECSM	**5.1**	**6.6**	**7.3**	**19.9**	**45.5**
MetaCS-FL_0.15	731.7	643.2	734.9	792.2	925.7
MetaCS-FL_0.25	788.2	751.0	732.5	831.8	1111.5
MetaCS-FL_0.5	912.3	968.2	978.5	1124.2	1383.9

ECSM consistently exhibited the lowest overhead across all client pools (5.1–45.5 s), closely followed by DivFL for smaller client pools and FedAvg for larger ones. The MetaCS-FL variants incurred higher overheads (643–1384 s), increasing with the number of clients but remaining substantially lower than MEC and ECMTC (1973–7653 s).

For the largest client pool (1000 clients), MetaCS-FL overhead ranged from 926–1384 s, while MEC and ECMTC reached 5871–7653 s.

Figure 27 and Table 26 present the total client selection overhead (in seconds) across different client pool sizes for the 40,000-task case.

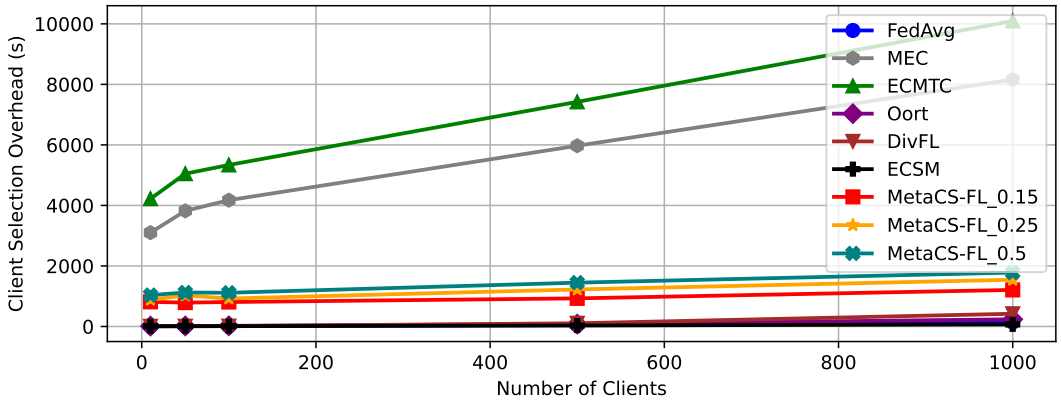

Figure 27: Cumulative selection overhead across all client pools over 100 simulated rounds (40,000 fixed workload).

Table 26: Cumulative selection overhead for 40,000 tasks with varying numbers of clients. Lowest cumulative overheads for each client count are highlighted in bold.

CS Approach	Number of Clients 10	50	100	500	1000
FedAvg	9.4	11.0	12.8	47.3	143.0
MEC	3101.1	3820.1	4173.6	5972.1	8161.6
ECMTC	4220.2	5048.7	5337.2	7422.1	10095.4
Oort	12.9	17.2	21.6	79.0	237.1
DivFL	7.3	9.8	13.0	106.1	420.5
ECSM	**7.1**	**8.8**	**10.8**	**31.1**	**70.9**
MetaCS-FL_0.15	814.6	788.7	807.8	928.9	1208.1
MetaCS-FL_0.25	885.5	1025.1	924.4	1224.1	1544.8
MetaCS-FL_0.5	1036.4	1122.8	1114.1	1448.2	1781.2

As with the 30,000-task case, ECSM achieved the lowest overhead across all client sizes (7.1–70.9 s), followed by DivFL for smaller pools (7.3–13.0 s) and FedAvg for larger pools (47.3–143.0 s). The MetaCS-FL variants incurred higher overhead (789–1781 s) and scaled efficiently with increasing client pool size. In contrast, MEC and ECMTC overhead grew sharply (3101–10095 s), reflecting limited scalability.

6.2.4.3 Discussion on workload and client pool scalability

FedAvg consistently exhibits low overhead for small client pools and remains competitive at larger scales, owing to its simple random selection, while Oort maintains low overhead for small to medium workloads but becomes less competitive at scale. DivFL and ECSM further improve scalability, with ECSM consistently achieving the lowest overhead across all workloads and client sizes, and DivFL performing competitively for smaller pools but becoming less efficient as client numbers increase.

MetaCS-FL incurs higher overhead for smaller workloads or client pools, which can be attributed to the client diversity criterion. When the workload is small, the diversity score often triggers new client selection. In contrast, for larger workloads and client pools, MetaCS-FL scales efficiently. This efficiency stems from three design choices: (i) the initial solution is generated using ECMTC, providing a strong starting point; (ii) previous solutions are reused when selection is not required, reducing redundant computations; and (iii) the metaheuristic is bounded by a 10-second runtime, limiting computational cost. Unlike MetaCS-FL, other algorithms perform client selection at every round, leading to a more direct increase in overhead as workload and client pool size grow.

Overall, MetaCS-FL provides scalable client selection while remaining adaptable to changes in client availability and workload size. The higher overhead observed for small workloads is mainly due to diversity-preserving awareness, which aims to balance client representation. However, these small cases do not reflect real-world cross-device FL deployments, where client pools and workloads are typically much larger, and the advantages of solution reuse and capped metaheuristic execution become more pronounced. In such large-scale scenarios, MetaCS-FL maintains stable and predictable overhead within the evaluated range of at most 1000 candidate clients and 40,000 tasks, while significantly outperforming MEC and ECMTC, although remaining more computationally demanding than lightweight approaches such as ECSM and FedAvg. The measurements do not establish scaling behavior beyond this range. Coarsening the task definition could reduce the task count and scheduling cost, but that option was not evaluated here and would sacrifice workload-allocation granularity.

6.2.5 Server-side Computational Overhead

To further answer **RQ5**, this subsection examines the server-side overhead of MetaCS-FL in terms of CPU time and memory usage under medium- and large-scale scenarios.

In addition to wall-clock selection time, we evaluate server-side CPU and memory overhead across medium (100 clients, 10,000 tasks) and large-scale (1000 clients, 40,000 tasks) scenarios. Table 27 reports the results, where CPU_{time} is the average process CPU time, $\text{RSS}_{\text{increase}}$ is the average increase in resident set size (RSS) during selection, and RSS_{peak} is the maximum observed RSS, all averaged over rounds. Process CPU time measures CPU seconds consumed by the selection process and is not a CPU-utilization percentage or a direct measurement of peak server load.

To further analyze the MetaCS-FL metaheuristic, Table 28 reports LNS-specific metrics (CPU time, memory usage, and number of iterations), averaged over LNS executions, *i.e.*, only when the metaheuristic is triggered. This differs from both the round-averaged values in Table 27 and the 100-round cumulative wall-clock totals in Section 6.2.4. When a new decision is triggered, selection precedes dispatch to the clients and its wall-clock latency is therefore part of the server-side coordination path, even though it is reported separately from the modeled client training makespan.

Table 27: Server-side computational overhead across representative scenarios.

CS Approach	**100 Clients, 10,000 Tasks**			**1000 Clients, 40,000 Tasks**		
	CPU_{time}	$\text{RSS}_{\text{increase}}$	RSS_{peak}	CPU_{time}	$\text{RSS}_{\text{increase}}$	RSS_{peak}
FedAvg	0.1	0.17	1.63	1.45	1.52	3.25
MEC	7.78	13.9	14.71	82.17	493.07	494.48
ECMTC	13.24	21.66	22.3	101.65	798.24	799.87
Oort	0.05	0.07	1.63	2.39	41.65	56.87
DivFL	0.05	**0.06**	**0.81**	4.24	18.97	21.42
ECSM	**0.04**	0.07	**0.81**	**0.71**	**11.58**	**12.18**
MetaCS-FL_0.15	4.85	2.27	22.93	12.17	72.68	797.98
MetaCS-FL_0.25	5.53	2.18	22.45	15.56	81.24	797.98
MetaCS-FL_0.5	6.75	2.61	22.93	17.94	81.58	797.37

Units: CPU_{time} in seconds; $\text{RSS}_{\text{increase}}$ and RSS_{peak} in megabytes.

ECSM, Oort, and FedAvg maintain low overhead, with ECSM achieving the lowest CPU usage. DivFL remains efficient, while MEC and ECMTC show substantially higher overhead that grows sharply with problem size.

MetaCS-FL incurs higher CPU usage due to the metaheuristic optimization, but its overhead grows in a controlled manner and remains well below MEC and ECMTC. Memory usage stays stable, with moderate increases and bounded peaks within the two measured scenarios. Given that per-core CPU-utilization traces were not recorded, these measurements characterize process cost and memory demand but cannot show instantaneous utilization peaks or contention with other server workloads.

The LNS-specific results indicate that the MetaCS-FL metaheuristic execution ac-

Table 28: LNS overhead within MetaCS-FL across representative scenarios.

CS Approach	100 Clients, 10,000 Tasks			1000 Clients, 40,000 Tasks		
	$\mathbf{CPU_{time}}$	$\mathbf{RSS_{increase}}$	Num. of Iterations	$\mathbf{CPU_{time}}$	$\mathbf{RSS_{increase}}$	Num. of Iterations
MetaCS-FL_0.15	**10.66**	0.81	**1185.21**	**11.70**	0.55	4.0
MetaCS-FL_0.25	10.68	**0.00**	1183.72	11.71	**0.47**	4.0
MetaCS-FL_0.5	10.75	0.78	1118.06	11.73	0.62	4.0

Units: CPU_{time} in seconds; $RSS_{increase}$ in megabytes.

counts for most of the computational cost, with stable CPU time (≈10–12 s) consistent with the imposed 10-second runtime budget and minimal memory usage. CPU seconds and wall-clock seconds are not identical measurements, particularly when numerical operations use multiple threads. The 10-second limit bounds the LNS refinement only and does not bound generation of the ECMTC initial solution, which explains why the exact scheduling component remains the principal scalability concern for large task counts.

A clear difference emerges in iterations: the medium-scale scenario (100 clients, 10,000 tasks) enables extensive exploration (over 1100 iterations), whereas the large-scale scenario (1000 clients, 40,000 tasks) yields only a few iterations per run due to higher iteration cost under a fixed runtime budget. This suggests adaptive runtime calibration or enforcing a minimum number of iterations may improve consistency across problem scales.

6.2.6 Sensitivity Analysis of MetaCS-FL Parameters

To answer **RQ6**, this subsection analyzes the sensitivity of MetaCS-FL to its main configuration parameters, including reselection triggers, objective-function weights, solution reuse, and metaheuristic search settings.

To assess the impact of MetaCS-FL parameters on the client-selection process, we conduct a sensitivity analysis using the same simulation-based setting as in the scalability experiments. Client performance metrics are synthetically generated from prior executions (Section 6.2.2). Each configuration is derived from the baseline setup (Section 6.1.7), with one parameter varied at a time and evaluated over three independent runs. This one-factor-at-a-time design isolates local parameter effects around the baseline and does not quantify all pairwise or higher-order interactions among the five objective weights, establish an optimal weight vector, or constitute an objective-ablation study. Table 29 summarizes the explored configurations.

For each configuration, we evaluate the number of metaheuristic executions (H),

Table 29: Parameter variations considered in the sensitivity analysis.

MetaCS-FL Component	**Parameter**	**Values**
Reselection Trigger	Training metrics Min. diversity score	$\{0.8, 0.85, 0.9\}$
	Max. diversity score decrease (%)	$\{5, 10, 20\}$
	Diversity history length (rounds)	$\{2, 3, 5\}$
	Max. makespan increase (%)	$\{5, 10, 20\}$
	Makespan history length (rounds)	$\{2, 3, 5\}$
	Max. energy increase (%)	$\{5, 10, 20\}$
	Energy history length (rounds)	$\{2, 3, 5\}$
	Test metrics Max. accuracy decrease (%)	$\{5, 10, 20\}$
	Accuracy history length (rounds)	$\{2, 3, 5\}$
Initial Solution Reuse	Reuse limit (rounds)	$\{5, 10, 20\}$
Objective Function	Objective function weights $(w_1, \ldots, w_5)$	Efficiency: $\{0.30,0.30,0.15,0.10,0.15\}$ Fairness: $\{0.05,0.05,0.45,0.30,0.15\}$ Utility: $\{0.05,0.05,0.15,0.15,0.60\}$
	α	$\{0.25, 0.35, 0.5\}$
	β	$\{0.5, 0.7, 0.9\}$
	q	$\{2, 3, 5\}$
Metaheuristic Optimization (LNS)	Time limit (s)	$\{1, 5, 10, 20\}$
	Max. number of iterations	$\{250, 500, 1000, 2000\}$
	Destroy fraction (%)	$\{10, 30, 50\}$
	Task removal range (min, max) (%)	Light: $(10, 50)$ Moderate: $(15, 75)$ Aggressive: $(25, 90)$
	Acceptance thresholds (%)	Conservative: $(0, 10, 10)$ Balanced: $(0.5, 25, 25)$ Exploratory: $(1, 40, 40)$

changes in the set of selected clients (S), average selected clients per round (C), and total selection time (T). We also report LNS statistics, including the number of candidate solutions (X_{cand}), accepted solutions (X_{acc}), and improved solutions (X_{imp}). Table 30 summarizes the most influential parameters by reporting the respective range of variation (Δ), computed as the difference between the maximum and minimum values observed across their tested configurations.

Table 30: Impact of the most influential parameters on MetaCS-FL execution.

Parameter	ΔH	ΔX_{cand}	ΔX_{acc}	ΔX_{imp}	ΔS	ΔT (s)
Min. diversity score	26.33	17,245.0	3,576.33	11.33	0.67	270.41
Max. diversity score decrease (%)	18.67	14,902.0	3,102.67	9.0	2.33	241.55
Max. number of iterations	1.33	73,748.67	12,568.0	6.0	3.67	1,089.93
Time limit (s)	9.0	51,264.33	9,328.0	22.0	9.67	753.68
Destroy fraction (%)	12.67	27,093.67	4,606.33	15.67	16.33	128.11
Task removal range (%)	10.33	31,880.67	6,742.33	13.67	8.67	211.74
Objective function weights	9.67	6,263.33	1,918.33	33.33	12.0	97.02
Max. makespan increase (%)	7.33	9,842.67	2,104.0	8.33	6.0	185.66
Max. energy increase (%)	6.67	8,921.33	1,955.67	7.67	5.67	172.48
Acceptance thresholds (%)	2.0	5,324.0	20,350.67	11.67	5.33	19.32
Reuse limit (rounds)	3.0	2,553.33	680.67	4.67	2.33	190.22

The results reveal distinct roles for each parameter group. Training diversity criteria most strongly affect reselection behavior, producing the largest variations in metaheuristic executions (ΔH) and major changes in total selection time, confirming their role as the main driver of responsiveness. In contrast, LNS stopping parameters dominate computational cost, yielding the largest variations in explored solutions (ΔX_{cand}) and runtime, thus controlling search depth and overhead.

Exploration-related parameters, more specifically the destroy fraction and task removal range, significantly affect both the number of explored solutions and variability in the selected client set (ΔS), highlighting their role in diversification. Objective-function weights primarily influence the observed search behavior, inducing the largest variation in improving solutions (ΔX_{imp}), while acceptance thresholds regulate filtering, strongly affecting accepted solutions (ΔX_{acc}) with limited impact on runtime. The variation in X_{imp} alone does not demonstrate end-to-end model quality, identify causal correlations among objectives, or prove that every objective is required.

System-level thresholds for makespan and energy introduce moderate variations, reflecting their role in triggering reselection under performance degradation. The reuse limit mainly affects recomputation frequency, with moderate impact on total selection time and limited influence on the search.

These results explain the configuration used in the main experiments. A diversity threshold of 0.85 balances responsiveness and stability, while limiting each LNS optimization run to 10 s balances solution refinement and computational overhead. The destroy configuration (30% of clients, 15%–75% task removal) ensures sufficient exploration, and the acceptance thresholds enforce controlled filtering. Finally, the objective weights em-

phasize utility and diversity, supporting effective refinement under constrained cost. They should, therefore, be interpreted as one controlled configuration motivated by the ECMTC starting point, not as a generally optimal prescription. A full factorial weight study, adaptive weighting across training stages, and ablation of individual objectives are necessary to characterize the trade-offs more completely.

6.3 Evaluation Under Dynamic Client Availability

In this set of experiments, we evaluate the behavior of the FL system under *dynamic client availability*. Specifically, we consider a system with a maximum of 100 clients, of which only a subset may participate at each communication round. Unlike the static setting, where the set of available clients remains unchanged throughout training, this scenario captures a more realistic deployment in which the pool of eligible clients evolves over time due to late arrivals, departures, or other factors affecting client availability.

We consider two cases of dynamic availability. The first is a *late-joining* scenario, in which training begins with fewer than 100 available clients and the remaining clients become eligible for participation after a predefined communication round. The second is an *intermittent-availability* scenario, in which all 100 clients are initially available, but client availability varies independently over time with individual clients temporarily becoming unavailable and later rejoining the system.

In both cases, client selection is restricted to the clients available in the communication round. This setup allows us to examine how each selection strategy adapts to a changing pool of eligible clients. Based on the static-setting results (Section 6.2.2), we focus on the non-IID configuration, as it represents the most challenging and informative case for assessing how dynamic availability interacts with statistical and system heterogeneity.

6.3.1 Late-Joining Clients

To answer **RQ7**, this subsection assesses how the number, arrival round, and profile of late-joining clients affect workload allocation, convergence, and system performance.

Following the experimental design of Bao *et al.* (17), we examine the effect of clients who become available only after training has started. While Bao *et al.* fixed the number of late-joining clients at 20 and varied only their entry round, we extend this setting by systematically varying three factors: (i) the number of late-joining clients, set to 10, 25, or

50; (ii) their joining round, set to 10, 25, or 50; and (iii) the system-performance category of the joining clients, classified as either *worst* or *best*.

Each client's performance category is derived from its system-level characteristics, including computation, network, and energy-related capabilities. These characteristics are aggregated into an overall client score, with the lowest- and highest-scoring clients labeled *worst* and *best*, respectively (see Appendix B.4). Thus, this categorization helps assess whether selection strategies can adapt to both changes in client availability and the delayed arrival of clients with different system capabilities.

For clarity, we report representative results for each dataset rather than the full factorial set. The selected tuples were chosen to cover contrasting late-joining conditions, including mild and delayed entry, small and large groups of joining clients, and favorable or adverse system-performance profiles. Since the impact of these factors differs across datasets, the reported tuples correspond to configurations that expose the main trends observed for each dataset. We evaluate the same performance metrics as in the static-availability experiments (see Section 6.1.8).

6.3.1.1 CIFAR-10 non-IID

Table 31 summarizes the late-joining performance on CIFAR-10 non-IID for the representative tuples (10,10,*best*) and (50,50,*worst*), which illustrate two contrasting cases: the early arrival of a small group of high-performance clients and the delayed arrival of a larger group of low-performance clients. Overall, we analyze four late-joining tuples: (10,10,*best*), (25,25,*worst*), (50,50,*best*), and (50,50,*worst*), all of which are discussed next.

For (10,10,*best*), MetaCS-FL reduces total training time from 58,460.30 s to 11,524.71 s (−80.29%) and total training energy from 5,121.23 kJ to 1,474.03 kJ (−71.22%) relative to FedAvg, requiring only 30 rounds on average to reach the target accuracy. FedCAB preserves the highest selection fairness among the client selection methods (0.82) but increases training time and energy by 15.19% and 3.65%, respectively, and reaches the target in 75 rounds. RIFLES increases training time and energy by 34.49% and 20.50%, respectively, and reaches the target in 91 rounds, but with lower fairness (0.46). In contrast, Oort, DivFL, and ECSM increase training time by 15.53%, 58.31%, and 50.36%, respectively, while increasing energy consumption by 19.74%, 36.43%, and 32.45%, respectively, with all three failing to reach the target accuracy.

For (25,25,*worst*), MetaCS-FL reduces total training time from 72,489.91 s to 15,594.36 s

Table 31: Late-joining performance for CIFAR-10 non-IID.

Tuple	CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.45
(10,10,*best*)	FedAvg (Baseline)	58,460.30	5,121.23	108.10	0.99	80.00 ± 0.82
	Oort	67,541.90 (+15.53%)	6,131.97 (+19.74%)	463.12 (+328.43%)	0.55 (−43.91%)	—
	DivFL	92,549.68 (+58.31%)	6,986.65 (+36.43%)	2,219.68 (+1953.41%)	0.41 (−58.82%)	—
	ECSM	87,902.50 (+50.36%)	6,783.00 (+32.45%)	358.08 (+231.26%)	0.53 (−46.22%)	—
	FedCAB	67,338.15 (+15.19%)	5,307.96 (+3.65%)	262.83 (+143.14%)	0.82 (−17.27%)	75.33 ± 5.25
	RIFLES	78,620.44 (+34.49%)	6,171.27 (+20.50%)	407.63 (+277.09%)	0.46 (−53.04%)	91.00 ± 6.68
	MetaCS-FL	11,524.71 (−80.29%)	1,474.03 (−71.22%)	318.96 (+195.07%)	0.71 (−28.57%)	30.33 ± 0.94
(50,50,*worst*)	FedAvg (Baseline)	69,605.88	5,231.02	102.71	0.96	84.33 ± 12.26
	Oort	59,472.96 (−14.56%)	5,477.82 (+4.72%)	372.31 (+262.49%)	0.67 (−30.23%)	87.67 ± 11.67
	DivFL	83,825.10 (+20.43%)	5,714.41 (+9.24%)	2,206.68 (+2048.49%)	0.40 (−58.07%)	97.00 ± 4.24
	ECSM	86,034.48 (+23.60%)	6,415.27 (+22.64%)	298.81 (+190.93%)	0.65 (−32.12%)	—
	FedCAB	84,701.74 (+21.69%)	6,377.36 (+21.91%)	286.67 (+179.11%)	0.77 (−19.09%)	97.00 ± 2.16
	RIFLES	75,966.29 (+9.14%)	5,785.39 (+10.60%)	312.59 (+204.34%)	0.50 (−47.53%)	86.67 ± 7.13
	MetaCS-FL	21,505.82 (−69.10%)	2,336.01 (−55.34%)	283.18 (+175.71%)	0.92 (−3.86%)	45.67 ± 4.03

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

(−78.49%) and total training energy from 5,611.73 kJ to 1,786.98 kJ (−68.16%) relative to FedAvg, requiring an average of 38 rounds to reach the target accuracy. FedCAB and RIFLES provide weaker trade-offs: FedCAB increases time and energy by 0.83% and 2.88%, respectively, while achieving the highest selection fairness among client selection methods (0.73). Meanwhile, RIFLES reduces training time by 7.15%, but increases energy by 5.54% and provides lower fairness (0.45). Oort reduces training time by 7.70%, but increases energy by 9.28% and requires more rounds than FedAvg. In contrast, DivFL and ECSM fail to reach the target accuracy while increasing training time by 18.58% and 18.98%, respectively, and energy by 5.38% and 12.88%.

For (50,50,*best*), MetaCS-FL reduces time from 59,054.46 s to 19,096.03 s (−67.66%) and energy from 4,916.49 kJ to 1,542.58 kJ (−68.62%) relative to FedAvg, requiring only 25 rounds on average to reach the target accuracy. FedCAB also provides substantial reductions, with 38,103.61 s, 3,084.14 kJ, 42 rounds, and high selection fairness (0.94). Oort and ECSM achieve smaller reductions in time and energy, with Oort reducing them

by 28.04% and 22.36%, and ECSM by 27.23% and 24.34%, respectively. ECSM also maintains high fairness (0.93) and reaches the target in 48 rounds. In contrast, DivFL provides limited reductions of 3.47% in time and 8.85% in energy while incurring a large selection overhead of 1,504.23 s, whereas RIFLES achieves smaller gains, reducing time and energy by 13.99% and 15.06%, respectively.

Finally, for (50,50,*worst*), MetaCS-FL reduces training time from 69,605.88 s to 21,505.82 s (−69.10%) and energy from 5,231.02 kJ to 2,336.01 kJ (−55.34%) relative to FedAvg, requiring 46 rounds on average to reach the target accuracy. FedCAB and RIFLES no longer provide cost reductions: FedCAB increases time and energy by 21.69% and 21.91%, respectively, while maintaining the highest selection fairness among client selection methods (0.77), whereas RIFLES increases time and energy by 9.14% and 10.60%, respectively, with lower fairness (0.50). Oort reduces time by 14.56%, but increases energy by 4.72% and requires more rounds than FedAvg. In contrast, DivFL and ECSM increase time by 20.43% and 23.60% and energy by 9.24% and 22.64%, respectively; ECSM fails to reach the target accuracy, whereas DivFL reaches it only after 97 rounds.

6.3.1.2 Fashion-MNIST non-IID

Table 32 summarizes the late-joining performance on Fashion-MNIST non-IID for the representative tuples (10,25,*worst*) and (50,50,*best*), which illustrate two contrasting cases: the adverse arrival of low-performance clients and the delayed arrival of a larger group of high-performance clients. Overall, we analyze four late-joining tuples: (10,25,*worst*), (25,50,*best*), (50,10,*best*), and (50,50,*best*), all of which are discussed next.

For (10,25,*worst*), MetaCS-FL reduces total training time from 9,257.63 s to 1,657.83 s (−82.09%) and total training energy from 804.76 kJ to 234.79 kJ (−70.83%) relative to FedAvg, requiring only 21 rounds on average to reach the target accuracy. Oort also provides substantial reductions, lowering training time and energy by 51.79% and 53.08%, respectively, and reaching the target in 27 rounds, but with lower fairness (0.54). FedCAB preserves the highest selection fairness among the client selection methods (0.85), while reducing time and energy by 23.50% and 30.43%, respectively, and reaching the target in 41 rounds. In contrast, DivFL and RIFLES increase training time by 36.08% and 40.70%, respectively, and energy by 22.59% and 35.35%, respectively. ECSM increases both costs more substantially, by 102.38% and 70.49%, and it fails to reach the target accuracy.

For (25,50,*best*), MetaCS-FL reduces total training time from 7,364.71 s to 1,924.21 s (−73.87%) and total training energy from 639.13 kJ to 285.94 kJ (−55.26%) relative to

Table 32: Late-joining performance for Fashion-MNIST non-IID.

Tuple	CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.7
(10,25,*worst*)	FedAvg (Baseline)	9,257.63	804.76	95.57	0.98	56.00 ± 9.90
	Oort	4,463.02 (−51.79%)	377.58 (−53.08%)	166.85 (+74.58%)	0.54 (−44.82%)	27.33 ± 5.31
	DivFL	12,598.01 (+36.08%)	986.54 (+22.59%)	932.21 (+875.41%)	0.41 (−57.94%)	70.00 ± 7.87
	ECSM	18,735.39 (+102.38%)	1,372.04 (+70.49%)	440.66 (+361.09%)	0.51 (−48.30%)	—
	FedCAB	7,082.03 (−23.50%)	559.90 (−30.43%)	185.77 (+94.38%)	0.85 (−13.85%)	41.00 ± 13.37
	RIFLES	13,025.29 (+40.70%)	1,089.29 (+35.35%)	452.77 (+373.76%)	0.46 (−53.62%)	84.00 ± 15.51
	MetaCS-FL	1,657.83 (−82.09%)	234.79 (−70.83%)	272.37 (+184.99%)	0.70 (−29.24%)	20.67 ± 0.94
(50,50,*best*)	FedAvg (Baseline)	11,127.40	971.64	98.12	0.90	61.00 ± 7.79
	Oort	9,446.20 (−15.11%)	842.40 (−13.30%)	242.80 (+147.46%)	0.91 (+1.67%)	51.00 ± 0.00
	DivFL	13,559.57 (+21.86%)	1,142.18 (+17.55%)	920.78 (+838.43%)	0.43 (−51.85%)	71.33 ± 20.29
	ECSM	19,235.31 (+72.86%)	1,601.26 (+64.80%)	372.82 (+279.97%)	0.61 (−32.18%)	—
	FedCAB	9,692.71 (−12.89%)	837.45 (−13.81%)	155.57 (+58.56%)	0.88 (−1.24%)	51.00 ± 0.00
	RIFLES	8,473.17 (−23.85%)	725.91 (−25.29%)	174.05 (+77.39%)	0.76 (−15.43%)	48.00 ± 5.66
	MetaCS-FL	3,063.44 (−72.47%)	296.45 (−69.49%)	156.37 (+59.36%)	0.94 (+5.33%)	19.33 ± 1.89

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

FedAvg, requiring an average of 20 rounds to reach the target accuracy. Oort provides smaller reductions, reducing time and energy by 11.97% and 4.05%, respectively, and reaches the target in 40 rounds. FedCAB achieves the highest selection fairness among client selection methods (0.89) but only slightly reduces time by 1.11% and slightly increases energy by 0.12%, reaching the target in 41 rounds. RIFLES increases training time and energy by 19.96% and 23.89%, respectively, with lower fairness (0.59). In contrast, DivFL and ECSM increase training time by 114.71% and 146.71%, respectively, and energy by 114.12% and 153.16%. ECSM also fails to reach the target accuracy.

For (50,10,*best*), MetaCS-FL reduces total training time from 8,085.58 s to 1,638.76 s (−79.73%) and total training energy from 710.00 kJ to 164.79 kJ (−76.79%) relative to FedAvg, requiring only 11 rounds on average to reach the target accuracy. Oort also reduces training time and energy by 21.85% and 24.81%, respectively, and reaches the target in 35 rounds, but with lower fairness (0.59). FedCAB slightly increases training time by 3.81%, while reducing energy by 3.77%, and reaches the target in 48 rounds with fair-

ness 0.78. RIFLES provides smaller reductions in time and energy, by 6.51% and 7.52%, respectively, and reaches the target in 46 rounds with fairness 0.71. In contrast, DivFL and ECSM increase training time by 131.58% and 122.76%, respectively, and energy by 118.57% and 107.91%, respectively, with both failing to reach the target accuracy.

Finally, for (50,50,*best*), MetaCS-FL reduces total training time from 11,127.40 s to 3,063.44 s (−72.47%) and total training energy from 971.64 kJ to 296.45 kJ (−69.49%) relative to FedAvg, requiring an average of 19 rounds to reach the target accuracy. RIFLES provides the strongest reductions among the other methods, decreasing time and energy by 23.85% and 25.29%, respectively, and reaching the target in 48 rounds, but with lower fairness (0.76). Oort and FedCAB achieve smaller reductions in time and energy, with Oort reducing them by 15.11% and 13.30%, and FedCAB by 12.89% and 13.81%, respectively; both reach the target in 51 rounds. In contrast, DivFL and ECSM increase training time by 21.86% and 72.86% and energy by 17.55% and 64.80%, respectively; ECSM fails to reach the target accuracy, whereas DivFL reaches it after 71 rounds.

6.3.1.3 Emotion non-IID

Table 33 summarizes the late-joining performance on Emotion non-IID for the representative tuples (10,50,*worst*) and (50,50,*best*), which illustrate two contrasting cases: the delayed arrival of a small group of low-performance clients and the delayed arrival of a larger group of high-performance clients. Overall, we analyze four late-joining tuples: (10,50,*worst*), (25,10,*best*), (50,25,*worst*), and (50,50,*best*), all of which are discussed next.

For (10,50,*worst*), MetaCS-FL reduces total training time from 32,458.69 s to 14,263.29 s (−56.06%) and total training energy from 1,723.39 kJ to 1,038.53 kJ (−39.74%) relative to FedAvg, requiring only 26 rounds on average to reach the target accuracy. RIFLES also provides substantial reductions, lowering training time and energy by 49.72% and 19.69%, respectively, and reaching the target in 26 rounds, but with lower fairness (0.56). FedCAB preserves the highest selection fairness among the client selection methods (0.91), while reducing time and energy by 19.79% and 19.90%, respectively, and reaching the target in 28 rounds. In contrast, Oort slightly reduces training time by 0.46%, but increases energy by 39.73% and reaches the target in 44 rounds. DivFL and ECSM increase training time by 61.24% and 158.65%, respectively, and energy by 30.32% and 123.53%, respectively.

For (25,10,*best*), MetaCS-FL reduces total training time from 29,577.65 s to 8,763.92 s (−70.37%) and total training energy from 1,914.27 kJ to 949.63 kJ (−50.39%) relative to FedAvg, requiring an average of 25 rounds to reach the target accuracy. Oort provides

Table 33: Late-joining performance for Emotion non-IID.

Tuple	CS Approach	M_{total}	Σ_{total}	T_{cs}	S_{fair}	RoA@0.8
(10,50,*worst*)	FedAvg (Baseline)	32,458.69	1,723.39	125.33	0.97	31.00 ± 0.82
	Oort	32,310.27 (−0.46%)	2,408.13 (+39.73%)	501.99 (+300.54%)	0.53 (−45.66%)	44.00 ± 2.83
	DivFL	52,336.92 (+61.24%)	2,245.91 (+30.32%)	1,765.56 (+1308.75%)	0.46 (−52.11%)	35.67 ± 2.87
	ECSM	83,955.02 (+158.65%)	3,852.21 (+123.53%)	726.42 (+479.61%)	0.49 (−49.01%)	91.33 ± 12.26
	FedCAB	26,034.10 (−19.79%)	1,380.42 (−19.90%)	235.57 (+87.96%)	0.91 (−5.97%)	28.33 ± 4.71
	RIFLES	16,320.90 (−49.72%)	1,384.04 (−19.69%)	235.29 (+87.74%)	0.56 (−41.85%)	26.00 ± 3.27
	MetaCS-FL	14,263.29 (−56.06%)	1,038.53 (−39.74%)	623.38 (+397.40%)	0.55 (−42.67%)	26.00 ± 2.16
(50,50,*best*)	FedAvg (Baseline)	42,806.85	2,177.80	120.64	0.99	31.67 ± 0.47
	Oort	38,387.86 (−10.32%)	2,322.62 (+6.65%)	236.40 (+95.96%)	0.77 (−22.26%)	31.67 ± 3.30
	DivFL	46,229.47 (+8.00%)	2,306.57 (+5.91%)	1,450.46 (+1102.32%)	0.78 (−21.45%)	30.33 ± 0.94
	ECSM	96,644.61 (+125.77%)	5,413.70 (+148.59%)	365.18 (+202.70%)	0.66 (−33.96%)	59.33 ± 28.77
	FedCAB	32,362.00 (−24.40%)	1,907.85 (−12.40%)	164.27 (+36.17%)	0.97 (−2.23%)	31.33 ± 7.13
	RIFLES	29,387.61 (−31.35%)	1,825.33 (−16.18%)	197.81 (+63.97%)	0.76 (−23.41%)	35.00 ± 11.58
	MetaCS-FL	20,612.34 (−51.85%)	1,388.21 (−36.26%)	246.03 (+103.94%)	0.87 (−12.13%)	20.67 ± 0.47

Units: M_{total} and T_{cs} in seconds; Σ_{total} in kilojoules.

smaller reductions, decreasing time and energy by 48.79% and 35.35%, respectively, and also reaches the target in 25 rounds. RIFLES reduces training time and energy by 32.03% and 18.60%, respectively, and reaches the target in 27 rounds with fairness 0.59. FedCAB increases training time by 12.03%, while reducing energy by 7.09%, and reaches the target in 28 rounds with the highest selection fairness among client selection methods (0.71). In contrast, DivFL and ECSM increase training time by 89.51% and 405.92%, respectively, and energy by 45.90% and 254.42%. ECSM also fails to reach the target accuracy.

For (50,25,*worst*), MetaCS-FL reduces total training time from 23,153.95 s to 10,717.82 s (−53.71%) and total training energy from 1,653.84 kJ to 1,073.56 kJ (−35.09%) relative to FedAvg, requiring only 22 rounds on average to reach the target accuracy. Oort reduces training time by 15.15%, but increases energy by 3.44% and reaches the target in 33 rounds with fairness 0.71. FedCAB slightly increases training time by 4.92%, while reducing energy by 3.56%, and reaches the target in 31 rounds with fairness 0.83. RIFLES increases training time by 2.84%, but slightly reduces energy by 0.60%, and reaches the

target in 31 rounds with lower fairness (0.51). In contrast, DivFL and ECSM increase training time by 85.60% and 531.76%, respectively, and energy by 49.25% and 318.82%, respectively; ECSM fails to reach the target accuracy.

Finally, for (50,50,*best*), MetaCS-FL reduces total training time from 42,806.85 s to 20,612.34 s (−51.85%) and total training energy from 2,177.80 kJ to 1,388.21 kJ (−36.26%) relative to FedAvg, requiring 21 rounds to reach the target accuracy. RIFLES provides the strongest reductions among the other methods, decreasing time and energy by 31.35% and 16.18%, respectively, and reaching the target in 35 rounds, but with lower fairness (0.76). FedCAB reduces time and energy by 24.40% and 12.40%, respectively, while maintaining the highest selection fairness among client selection methods (0.97) and reaching the target in 31 rounds. Oort reduces time by 10.32%, but increases energy by 6.65% and reaches the target in 32 rounds. In contrast, DivFL and ECSM increase training time by 8.00% and 125.77% and energy by 5.91% and 148.59%, respectively.

6.3.1.4 Late-client participation and workload allocation

Beyond aggregate performance, we further analyze how each selection strategy distributes the workload after late-joining clients become available. We focus on CIFAR-10 non-IID and on the two configurations in which all seven approaches have a post-entry training window before reaching the target accuracy: (10,10,*best*) and (25,25,*worst*). These settings capture two complementary cases: the early arrival of high-performance clients and the later arrival of low-performance clients. For each approach, we report the average post-entry shares of selections and scheduled samples assigned to initial and late-joining clients. These shares are computed within each approach rather than relative to FedAvg.

Table 34 summarizes how each approach selects clients and distributes samples after late-client entry, while Figure 28 highlights the corresponding scheduled-sample shares.

In the high-performance early-arrival configuration (10,10,*best*), late clients represent 10% of the population. FedAvg and FedCAB remain close to proportional use, selecting 9.73% and 11.38% of clients and assigning 9.70% and 10.22% of samples, respectively, to late joiners. In contrast, MetaCS-FL over-represents these high-performance late clients, selecting 17.81% of clients and assigning 18.05% of samples, indicating that it reallocates workload toward newly available clients when they improve the system-level schedule. This is consistent with its aggregate performance, as MetaCS-FL reaches RoA@0.45 much earlier and reduces total training time and energy by 80.29% and 71.22%, respectively, relative to FedAvg. Oort and RIFLES also increase late-client use, selecting 14.95% and

Table 34: Late-joining workload allocation for CIFAR-10 non-IID.

Tuple	CS Approach	Number of Selected Clients		Number of Scheduled Samples	
		Initial	Late-Joining	Initial	Late-Joining
(10,10,*best*)	FedAvg	44.18 (90.27%)	4.76 (9.73%)	18,060.27 (90.30%)	1,939.73 (9.70%)
	Oort	33.10 (85.05%)	5.82 (14.95%)	17,126.73 (85.63%)	2,873.27 (14.37%)
	DivFL	40.67 (100.00%)	0.00 (0.00%)	20,000.00 (100.00%)	0.00 (0.00%)
	ECSM	37.35 (91.46%)	3.54 (8.54%)	18,244.37 (91.22%)	1,755.63 (8.78%)
	FedCAB	36.59 (88.62%)	4.70 (11.38%)	17,470.60 (89.78%)	1,988.36 (10.22%)
	RIFLES	35.52 (86.66%)	5.47 (13.34%)	16,856.20 (87.13%)	2,490.00 (12.87%)
	MetaCS-FL	39.18 (82.19%)	8.49 (17.81%)	16,216.80 (81.95%)	3,570.28 (18.05%)
(25,25,*worst*)	FedAvg	36.85 (75.40%)	12.02 (24.60%)	15,077.76 (75.39%)	4,922.24 (24.61%)
	Oort	27.36 (70.30%)	11.56 (29.70%)	13,958.44 (69.79%)	6,041.56 (30.21%)
	DivFL	40.67 (100.00%)	0.00 (0.00%)	20,000.00 (100.00%)	0.00 (0.00%)
	ECSM	32.38 (78.34%)	8.98 (21.66%)	15,708.18 (78.54%)	4,291.82 (21.46%)
	FedCAB	23.24 (56.34%)	18.01 (43.66%)	11,169.87 (56.48%)	8,618.74 (43.52%)
	RIFLES	40.05 (98.43%)	0.64 (1.57%)	19,488.19 (98.55%)	284.24 (1.45%)
	MetaCS-FL	36.17 (84.61%)	6.70 (15.39%)	16,299.47 (84.41%)	3,013.43 (15.59%)

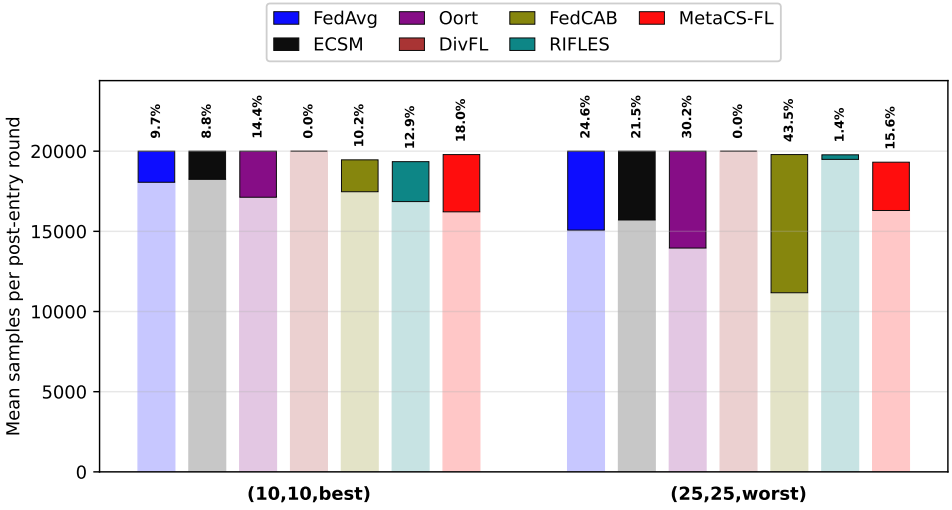

Figure 28: Average scheduled-sample distribution between initial and late-joining clients after their entry on CIFAR-10 non-IID, grouped by late-joining configuration.

13.34% of clients, respectively, consistent with Oort's utility-based exploration and RIFLES' expected-time-based eligibility rule. ECSM slightly underuses late clients, selecting 8.54% of clients and assigning 8.78% of samples to them, reflecting its dependence on his-

torical accuracy and reputation. DivFL assigns no workload to late clients, suggesting that they do not improve its diversity-oriented representative set.

In the low-performance late-arrival configuration (25,25,*worst*), late clients represent 25% of the population. FedAvg remains close to proportional use, selecting 24.60% of clients and assigning 24.61% of samples to late joiners. Oort over-represents them, selecting 29.70% of clients and assigning 30.21% of samples, indicating that its exploration and utility mechanisms can still promote system-poor late clients, which explains its lower training time but higher energy consumption relative to FedAvg. FedCAB shows the strongest shift toward late joiners, selecting 43.66% of clients and assigning 43.52% of samples to them, reflecting its participation-compensation mechanism. However, this inclusion does not improve efficiency because these clients have unfavorable system characteristics. In contrast, RIFLES nearly excludes late clients, selecting only 1.57% of clients and assigning 1.45% of samples to them, consistent with its expected-time feasibility criterion. MetaCS-FL adopts an intermediate behavior, selecting 15.39% of clients and assigning 15.59% of samples to late joiners, limiting their workload without excluding them entirely while still achieving the largest reductions in total training time and energy, by 78.49% and 68.16%, respectively, relative to FedAvg.

6.3.1.5 Discussion on late-joining scenario

FedAvg incorporates late clients mainly according to availability, without explicitly exploiting high-performance arrivals or avoiding low-performance ones. Oort may explore late clients through its exploration mechanism, which can be beneficial when new clients are resource-efficient. However, it can also increase energy consumption or reduce fairness when these clients have poor device capabilities. DivFL benefits from late clients only when they improve diversity representation, whereas ECSM tends to be conservative because late clients lack sufficient reputation and accuracy history, making it less robust when the client pool changes during training.

FedCAB and RIFLES adapt more explicitly to dynamic participation, but with different trade-offs. FedCAB compensates late and under-participating clients, which usually improves selection fairness; however, this can increase the workload assigned to poor devices when low-performance clients arrive late. RIFLES emphasizes availability and expected execution-time feasibility, helping it avoid clients that may delay the round, but this filtering can repeatedly favor a smaller subset of feasible clients and does not consistently optimize energy or fairness.

MetaCS-FL shows the most consistent behavior across the late-joining scenarios because it adapts client workload according to multiple criteria rather than a single selection signal. By considering the expected effects on makespan, energy, diversity, and utility, it can exploit high-performance late joiners when they improve the training process and limit the participation of low-performance late joiners when they would increase costs. This schedule-aware adaptation explains why MetaCS-FL achieves the largest and most consistent reductions in total training time and energy while requiring fewer rounds to reach the target accuracy in most scenarios.

6.3.2 Intermittent-Availability Clients

To answer **RQ8**, this subsection evaluates the robustness of MetaCS-FL when clients become temporarily unavailable, later return, or fail to complete assigned workloads under intermittent-availability conditions.

The second dynamic-availability scenario evaluates the selection strategies under intermittent client participation while also considering the non-IID configuration. In contrast to the late-joining scenario, all 100 clients are initially part of the FL system, but their availability may change across communication rounds. A client may temporarily become unavailable, later return, or fail to complete an assigned workload after selection.

Following prior work that models client availability in FL as a Markovian stochastic process over communication rounds (18), and more broadly studies heterogeneous and time-varying client unavailability (20), we employ a two-state Markov availability process to simulate intermittent participation. The process, described in detail in Appendix B.5, assigns clients to four availability profiles: *stable*, *intermittent*, *unstable*, and *bursty-off*. These profiles represent different degrees of availability and reliability over time, allowing the experiment to capture both transient unavailability and longer-term differences in client behavior. The model also accounts for post-selection failures, in which a selected client does not complete its assigned workload. The transition and completion-failure processes are controlled emulations rather than models calibrated from production device-disconnection or preemptible-instance traces. They support reproducible stress testing, but conclusions about real deployment frequencies require trace-based validation.

We consider two intermittent-availability settings, denoted as *moderate* and *severe*. The *moderate* setting models intermittent availability with a larger share of reliable clients and lower completion-failure probability, whereas the *severe* setting shifts the population toward less reliable clients and increases the likelihood of post-selection failures. Table 35

summarizes the proportions and completion-failure parameters used in each setting.

Table 35: Client-profile composition and failure settings for intermittent availability.

Client Profile	**Moderate**	**Severe**
Stable clients	25%	10%
Intermittent clients	50%	40%
Unstable clients	20%	35%
Bursty-off clients	5%	15%
Completion-failure probability	Lower	Higher
Maximum completion-failure probability	0.30	0.35

All approaches schedule exactly the configured workload in these experiments. No speculative duplicate tasks or backup-client over-assignment is used. Consequently, the results evaluate reliability-aware workload restriction and redistribution, not redundancy-based straggler mitigation.

In addition to the performance metrics used in the previous experiments, we assess availability-specific metrics that characterize both client-level failures and workload-level failure impact. Specifically, $\bar{n}_{\text{sel}}$ is the mean number of selected clients per round, $\bar{n}_{\text{fail}}$ is the mean number of failing selected clients, and $\bar{n}_{\text{fail}}/\bar{n}_{\text{sel}}$ is the corresponding failure rate among selected clients. At the workload level, $\bar{s}_{\text{fail}}$ denotes the mean number of failed scheduled samples, $\bar{s}_{\text{fail}}/\bar{n}_{\text{fail}}$ measures the failed samples per failing client, and $\bar{s}_{\text{fail}}/\bar{s}_{\text{sched}}$ measures the fraction of scheduled samples that fail to be processed. For brevity, the tables presented in the following analyses focus only on these availability-specific metrics.

6.3.2.1 CIFAR-10 non-IID

Table 36 reports the average client-level failure and workload-level impact metrics for CIFAR-10 non-IID under the *moderate* and *severe* intermittent-availability scenarios. The corresponding test accuracy progressions are shown in Figure 29.

Across both intermittent-availability scenarios, MetaCS-FL achieves the best overall performance. In the *moderate* case, it reduces total training time from 52,377.23 s to 17,005.17 s (−67.53%) and total training energy from 3,965.54 kJ to 1,897.44 kJ (−52.15%) relative to FedAvg, while reaching the target accuracy in 45 rounds. By comparison, Oort, FedCAB, and RIFLES require 56,017.15 s, 66,379.28 s, and 73,941.21 s, consume 4,515.39 kJ, 4,944.37 kJ, and 5,643.02 kJ, and reach the target only after 73, 79, and 89 rounds, respectively. DivFL and ECSM require 93,628.86 s and 85,484.14 s, consume 6,618.45 kJ and 6,463.13 kJ, and do not reach the target within 100 rounds.

Table 36: Client failures and failed scheduled samples for CIFAR-10 non-IID.

Intermittent Availability	CS Approach	$\bar{n}_{\text{sel}}$	$\bar{n}_{\text{fail}}$	$\bar{n}_{\text{fail}}/\bar{n}_{\text{sel}}$ (%)	$\bar{s}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{n}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{s}_{\text{sched}}$ (%)
Moderate	FedAvg (Baseline)	41.29	2.48	6.01	1,190.42	479.52	5.95
	Oort	41.40 (+0.25%)	1.92 (−22.80%)	4.63 (−22.99%)	904.83 (−23.99%)	471.85 (−1.60%)	4.52 (−23.99%)
	DivFL	36.55 (−11.49%)	1.90 (−23.34%)	5.21 (−13.39%)	1,056.89 (−11.22%)	555.29 (+15.80%)	5.28 (−11.22%)
	ECSM	42.10 (+1.95%)	2.31 (−6.96%)	5.49 (−8.73%)	1,102.69 (−7.37%)	477.12 (−0.50%)	5.51 (−7.37%)
	FedCAB	41.15 (−0.35%)	2.86 (+15.11%)	6.95 (+15.52%)	1,387.35 (+16.54%)	485.43 (+1.23%)	6.94 (+16.54%)
	RIFLES	41.20 (−0.22%)	2.49 (+0.40%)	6.05 (+0.63%)	1,221.11 (+2.58%)	489.82 (+2.15%)	6.11 (+2.58%)
	MetaCS-FL	43.96 (+6.47%)	2.28 (−8.25%)	5.18 (−13.82%)	1,007.32 (−15.38%)	442.22 (−7.78%)	5.04 (−15.38%)
Severe	FedAvg (Baseline)	40.58	4.19	10.32	2,061.00	492.27	10.30
	Oort	41.10 (+1.29%)	3.65 (−12.74%)	8.89 (−13.83%)	1,748.28 (−15.17%)	478.44 (−2.81%)	8.74 (−15.17%)
	DivFL	36.99 (−8.85%)	3.60 (−14.12%)	9.72 (−5.78%)	1,955.51 (−5.12%)	543.86 (+10.48%)	9.78 (−5.12%)
	ECSM	41.26 (+1.66%)	4.05 (−3.31%)	9.82 (−4.85%)	1,976.81 (−4.08%)	488.37 (−0.79%)	9.88 (−4.08%)
	FedCAB	41.05 (+1.14%)	4.84 (+15.62%)	11.79 (+14.32%)	2,351.03 (+14.07%)	485.64 (−1.35%)	11.76 (+14.07%)
	RIFLES	41.25 (+1.64%)	4.31 (+2.90%)	10.44 (+1.24%)	2,116.09 (+2.67%)	491.15 (−0.23%)	10.58 (+2.67%)
	MetaCS-FL	46.99 (+15.80%)	4.48 (+7.05%)	9.54 (−7.53%)	1,863.68 (−9.57%)	415.83 (−15.53%)	9.32 (−9.57%)

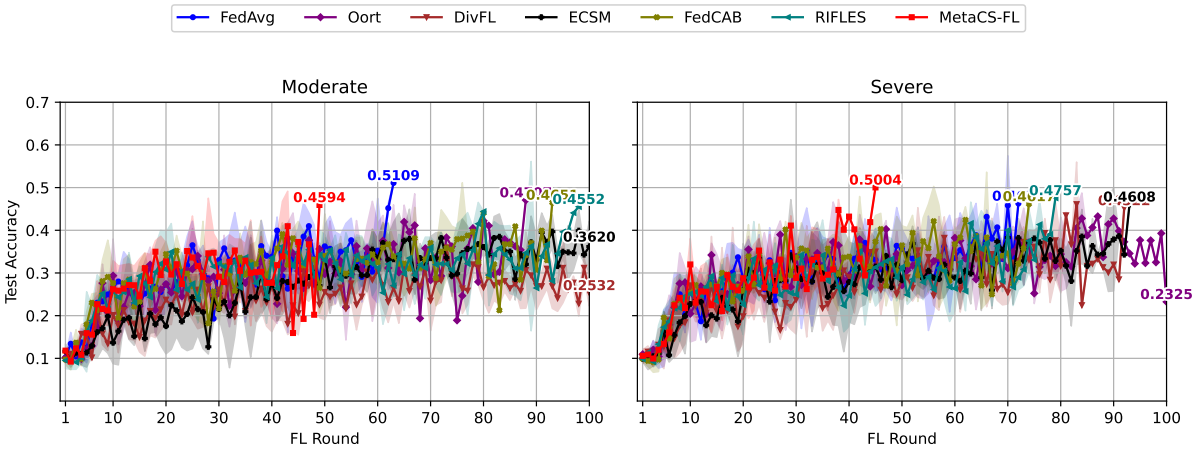

Figure 29: Test accuracy progression for the 100-client system on CIFAR-10 non-IID under *moderate* and *severe* intermittent availability.

In the *severe* case, the corresponding reductions of MetaCS-FL are from 59,102.78 s to 17,586.41 s (−70.24%) and from 4,276.85 kJ to 1,976.71 kJ (−53.78%), with convergence in 40 rounds. The remaining approaches require between 56,113.93 s and 75,463.11 s, consume between 4,069.18 kJ and 5,501.89 kJ, and reach the target after 65 to 89 rounds.

MetaCS-FL does not mitigate failures through conservative client selection. Instead, it selects the largest client set among the evaluated methods in both scenarios. In the *moderate* case, it reduces the mean number of failing clients from 2.48 to 2.28 and lowers the client-level failure rate from 6.01% to 5.18%, compared to FedAvg. In the *severe* case, the mean number of failing clients increases from 4.19 to 4.48 because more clients are selected, but the failure rate still decreases from 10.32% to 9.54%.

At the workload level, MetaCS-FL consistently reduces the work assigned to clients that eventually fail. In the *moderate* case, it reduces failed scheduled samples from 1,190.42 to 1,007.32 and lowers the failed samples per failing client from 479.52 to 442.22. In the *severe* case, these values decrease from 2,061.00 to 1,863.68 and from 492.27 to 415.83, respectively. Thus, even when failures occur, failed clients under MetaCS-FL tend to carry smaller workloads. Although Oort reports fewer raw failing clients and a lower failed-sample fraction, it does so with fewer selected clients and worse time, energy, and round-to-accuracy performance.

6.3.2.2 Fashion-MNIST non-IID

Table 37 reports the average client-level failure and workload-level impact metrics for Fashion-MNIST non-IID under the *moderate* and *severe* intermittent-availability scenarios. The corresponding test accuracy progressions are shown in Figure 30.

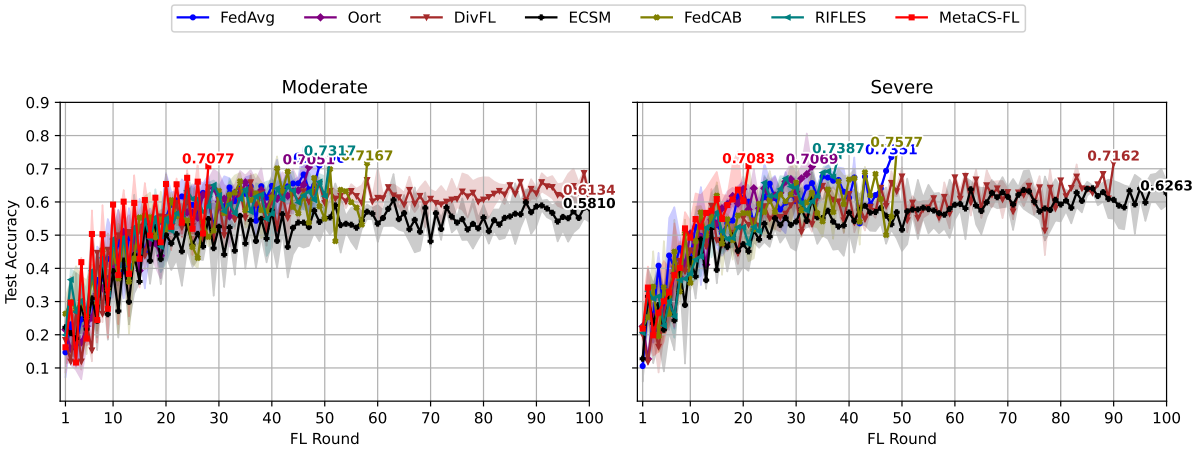

Figure 30: Test accuracy progression for the 100-client system on Fashion-MNIST non-IID under *moderate* and *severe* intermittent availability.

Across both intermittent-availability scenarios, MetaCS-FL achieves the best overall performance. In the *moderate* case, it reduces total training time from 8,163.69 s to 2,413.58 s (−70.44%) and total training energy from 641.08 kJ to 261.19 kJ (−59.26%)

Table 37: Client failures and failed scheduled samples for Fashion-MNIST non-IID.

Intermittent Availability	CS Approach	$\bar{n}_{\text{sel}}$	$\bar{n}_{\text{fail}}$	$\bar{n}_{\text{fail}}/\bar{n}_{\text{sel}}$ (%)	$\bar{s}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{n}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{s}_{\text{sched}}$ (%)
Moderate	FedAvg (Baseline)	41.59	2.50	6.02	1,433.13	572.41	5.97
	Oort	41.56 (−0.06%)	1.94 (−22.51%)	4.67 (−22.48%)	1,081.67 (−24.52%)	557.51 (−2.60%)	4.51 (−24.52%)
	DivFL	36.54 (−12.14%)	1.93 (−22.78%)	5.29 (−12.11%)	1,285.92 (−10.27%)	665.14 (+16.20%)	5.36 (−10.27%)
	ECSM	39.59 (−4.80%)	2.23 (−11.06%)	5.63 (−6.55%)	1,345.26 (−6.13%)	604.19 (+5.55%)	5.61 (−6.13%)
	FedCAB	41.07 (−1.23%)	2.90 (+15.68%)	7.05 (+17.13%)	1,683.18 (+17.45%)	581.17 (+1.53%)	7.01 (+17.45%)
	RIFLES	41.11 (−1.14%)	2.54 (+1.56%)	6.18 (+2.73%)	1,496.80 (+4.44%)	588.68 (+2.84%)	6.24 (+4.44%)
	MetaCS-FL	42.89 (+3.12%)	1.98 (−20.77%)	4.63 (−23.15%)	1,110.76 (−22.49%)	560.56 (−2.07%)	4.63 (−22.49%)
Severe	FedAvg (Baseline)	40.81	4.42	10.84	2,572.53	581.69	10.72
	Oort	42.32 (+3.69%)	3.55 (−19.65%)	8.40 (−22.51%)	1,986.64 (−22.77%)	559.11 (−3.88%)	8.28 (−22.77%)
	DivFL	37.01 (−9.31%)	3.68 (−16.78%)	9.95 (−8.23%)	2,400.47 (−6.69%)	652.15 (+12.11%)	10.00 (−6.69%)
	ECSM	40.71 (−0.24%)	4.14 (−6.47%)	10.16 (−6.26%)	2,456.75 (−4.50%)	594.05 (+2.13%)	10.24 (−4.50%)
	FedCAB	41.14 (+0.81%)	5.00 (+13.14%)	12.16 (+12.23%)	2,896.51 (+12.59%)	578.84 (−0.49%)	12.07 (+12.59%)
	RIFLES	41.13 (+0.78%)	4.51 (+1.93%)	10.96 (+1.15%)	2,665.54 (+3.62%)	591.22 (+1.64%)	11.11 (+3.62%)
	MetaCS-FL	45.03 (+10.35%)	4.17 (−5.72%)	9.26 (−14.56%)	2,196.70 (−14.61%)	526.73 (−9.45%)	9.15 (−14.61%)

relative to FedAvg, while reaching the target accuracy in 26 rounds. Oort, FedCAB, and RIFLES require 7,490.54 s, 8,166.24 s, and 8,650.53 s, consume 623.82 kJ, 619.88 kJ, and 677.07 kJ, and reach the target after 46, 47, and 51 rounds, respectively. DivFL and ECSM require 17,760.38 s and 17,556.43 s, consume 1,276.35 kJ and 1,305.42 kJ, and do not reach the target within 100 rounds. In the *severe* case, the corresponding reductions of MetaCS-FL are from 7,033.60 s to 1,660.41 s (−76.39%) and from 546.47 kJ to 203.06 kJ (−62.84%), with convergence in 19 rounds. The remaining approaches require between 5,231.82 s and 17,149.18 s, consume between 436.23 kJ and 1,230.90 kJ, and either reach the target after 32 to 84 rounds or, for ECSM, do not reach it within 100 rounds.

MetaCS-FL preserves broad participation rather than mitigating failures through conservative client selection. It selects the largest client set among the evaluated methods in both scenarios, with 42.89 clients per round in the *moderate* case and 45.03 in the *severe* case. Even with this larger selected-client set, it reduces the mean number of failing clients from 2.50 to 1.98 and lowers the client-level failure rate from 6.02% to 4.63% in the *moderate* case, compared to FedAvg. In the *severe* case, it also reduces the mean number of failing clients from 4.42 to 4.17 and decreases the failure rate from 10.84% to 9.26%.

This behavior is also reflected in the workload-level metrics. In the *moderate* case, MetaCS-FL reduces failed scheduled samples from 1,433.13 to 1,110.76 and lowers the failed samples per failing client from 572.41 to 560.56, compared to FedAvg. In the *severe* case, these values decrease from 2,572.53 to 2,196.70 and from 581.69 to 526.73, respectively. Thus, failed clients under MetaCS-FL tend to carry smaller workloads even when the selected-client set is larger. Although Oort reports fewer raw failing clients and a lower failed-sample fraction, its time, energy, and round-to-accuracy performance remain worse in both scenarios.

6.3.2.3 Emotion non-IID

Table 38 reports the average client-level failure and workload-level impact metrics for Emotion non-IID under the *moderate* and *severe* intermittent-availability scenarios. The corresponding test accuracy progressions are shown in Figure 31.

Table 38: Client failures and failed scheduled samples for Emotion non-IID.

Intermittent Availability	CS Approach	$\bar{n}_{\text{sel}}$	$\bar{n}_{\text{fail}}$	$\bar{n}_{\text{fail}}/\bar{n}_{\text{sel}}$ (%)	$\bar{s}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{n}_{\text{fail}}$	$\bar{s}_{\text{fail}}/\bar{s}_{\text{sched}}$ (%)
Moderate	FedAvg (Baseline)	42.06	2.46	5.84	7,576.65	3,082.00	5.68
	Oort	41.45 (−1.45%)	2.23 (−9.08%)	5.38 (−7.81%)	6,946.29 (−8.32%)	3,116.86 (+1.13%)	5.21 (−8.32%)
	DivFL	29.76 (−29.26%)	1.59 (−35.17%)	5.35 (−8.37%)	7,200.72 (−4.96%)	4,521.33 (+46.70%)	5.40 (−4.96%)
	ECSM	39.39 (−6.35%)	2.28 (−7.23%)	5.79 (−0.86%)	7,610.25 (+0.44%)	3,340.12 (+8.37%)	5.70 (+0.44%)
	FedCAB	37.79 (−10.15%)	2.59 (+5.32%)	6.85 (+17.23%)	8,851.89 (+16.83%)	3,421.62 (+11.02%)	6.63 (+16.83%)
	RIFLES	42.23 (+0.40%)	2.27 (−7.45%)	5.38 (−7.81%)	6,663.79 (−12.05%)	2,929.33 (−4.95%)	4.99 (−12.05%)
	MetaCS-FL	41.20 (−2.05%)	2.26 (−7.94%)	5.49 (−6.04%)	6,643.03 (−12.32%)	2,937.32 (−4.69%)	4.98 (−12.30%)
Severe	FedAvg (Baseline)	41.31	4.15	10.04	13,082.32	3,153.83	9.80
	Oort	41.46 (+0.38%)	3.90 (−5.91%)	9.42 (−6.19%)	12,062.36 (−7.80%)	3,089.04 (−2.05%)	9.04 (−7.80%)
	DivFL	29.87 (−27.69%)	3.01 (−27.40%)	10.08 (+0.42%)	13,581.76 (+3.82%)	4,511.07 (+43.03%)	10.18 (+3.82%)
	ECSM	34.74 (−15.90%)	3.73 (−9.95%)	10.75 (+7.13%)	14,200.09 (+8.54%)	3,805.57 (+20.66%)	10.64 (+8.54%)
	FedCAB	38.28 (−7.33%)	4.57 (+10.27%)	11.95 (+19.01%)	15,516.53 (+18.61%)	3,393.14 (+7.59%)	11.63 (+18.61%)
	RIFLES	42.72 (+3.41%)	4.42 (+6.55%)	10.34 (+3.06%)	12,824.45 (−1.97%)	2,901.96 (−7.99%)	9.61 (−1.97%)
	MetaCS-FL	41.42 (+0.28%)	3.98 (−4.12%)	9.60 (−4.39%)	11,919.22 (−8.89%)	2,999.88 (−4.88%)	8.93 (−8.86%)

Across both intermittent-availability scenarios, MetaCS-FL achieves the best overall performance. In the *moderate* case, it reduces total training time from 33,829.46 s to 6,908.61 s (−79.58%) and total training energy from 1,850.40 kJ to 698.98 kJ (−62.23%)

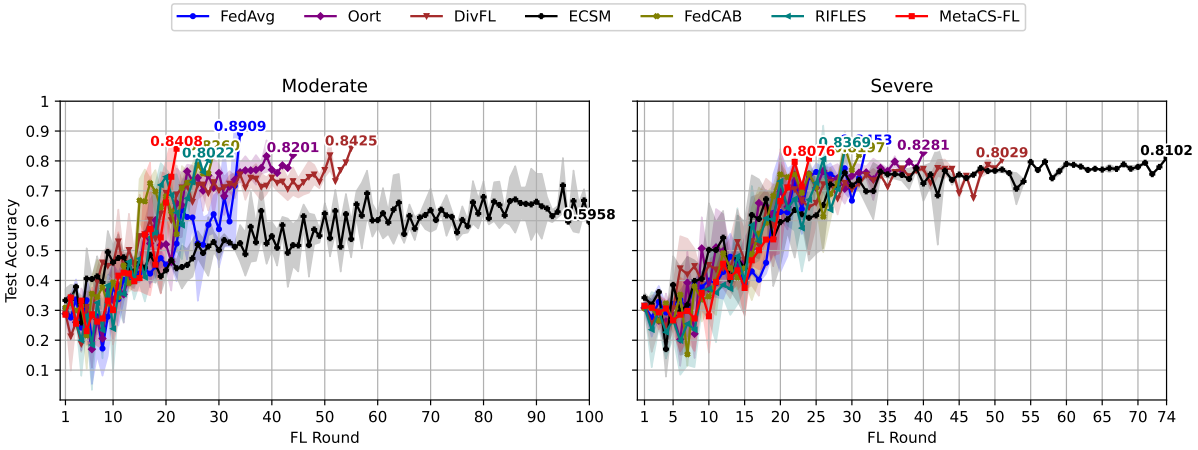

Figure 31: Test accuracy progression for the 100-client system on Emotion non-IID under *moderate* and *severe* intermittent availability.

relative to FedAvg, while reaching the target accuracy in 20 rounds. By comparison, Oort, FedCAB, and RIFLES require 29,486.41 s, 29,511.10 s, and 16,878.11 s, consume 1,779.22 kJ, 1,632.20 kJ, and 1,313.52 kJ, and reach the target after 40, 27, and 27 rounds, respectively. DivFL and ECSM require 56,967.01 s and 112,918.50 s, consume 3,218.08 kJ and 6,277.50 kJ, and reach the target only after 52 and 98 rounds. In the *severe* case, the corresponding reductions of MetaCS-FL are from 30,215.79 s to 10,179.63 s (−66.31%) and from 1,648.11 kJ to 813.47 kJ (−50.64%), with convergence in 23 rounds. The remaining approaches require between 18,262.66 s and 77,710.01 s, consume between 1,329.07 kJ and 4,307.15 kJ, and reach the target after 27 to 74 rounds.

MetaCS-FL also mitigates failures without strongly reducing the number of selected clients. In the *moderate* case, it keeps the number of selected clients close to FedAvg, with 41.20 clients per round compared to 42.06, while reducing the mean number of failing clients from 2.46 to 2.26 and lowering the client-level failure rate from 5.84% to 5.49%. In the *severe* case, MetaCS-FL selects slightly more clients than FedAvg, 41.42 compared to 41.31, while reducing the mean number of failing clients from 4.15 to 3.98 and decreasing the failure rate from 10.04% to 9.60%.

At the workload level, MetaCS-FL reduces the amount of assigned data lost under intermittent availability. In the *moderate* case, the number of failed scheduled samples decreases from 7,576.65 to 6,643.04, while the fraction of scheduled samples that fail to be processed decreases from 5.68% to 4.98%, compared to FedAvg. In the *severe* case, the corresponding reductions are from 13,082.32 to 11,919.22 and from 9.80% to 8.93%, respectively. The number of failed samples per failing client also decreases from 3,082.00 to 2,937.32 in the *moderate* case and from 3,153.83 to 2,999.88 in the *severe*

case. Although RIFLES reports a slightly lower value for this per-client metric, MetaCS-FL achieves a lower total amount of failed assigned data and the best time, energy, and round-to-accuracy performance across both scenarios.

6.3.2.4 Discussion on intermittent-availability scenario

FedAvg does not explicitly account for intermittent availability and therefore selects clients according to the available pool without considering their likelihood of failing after selection. As a result, it can assign workloads to clients that later fail to complete execution, increasing the number of failed scheduled samples. Oort generally reduces the number of failing clients and the failed-sample fraction by favoring more reliable or higher-utility clients, but this behavior is often associated with a smaller selected-client set and does not consistently translate into better time, energy, or round-to-accuracy performance. DivFL and ECSM also reduce failures in some cases, but their selection criteria are not designed to jointly control execution efficiency and workload loss under post-selection failures.

FedCAB and RIFLES handle dynamic participation more directly, but with different limitations. FedCAB encourages participation from under-selected clients, which can improve fairness but may also increase failures when less reliable clients are selected. RIFLES uses availability-aware scheduling to avoid clients expected to be unavailable or slow, which can reduce failed scheduled samples in some settings. However, this filtering may restrict participation to a narrower set of feasible clients and does not minimize total training time or energy across datasets and intermittent-availability settings.

MetaCS-FL shows the most consistent behavior under intermittent availability through its multi-criteria adaptation of client selection and assigned workload. It does not mitigate failures by simply selecting fewer clients. Instead, it uses past availability and completion outcomes to restrict the admissible workload capacities of less reliable clients. As a result, less reliable clients remain eligible for selection, but their potential workload assignments are limited. This helps explain why MetaCS-FL achieves the best overall time, energy, and round-to-accuracy performance across the intermittent-availability scenarios, while limiting the number of scheduled samples that fail to be processed after selection.

6.4 General Discussion

The experimental results show that MetaCS-FL provides the most balanced trade-off among system efficiency, learning progress, fairness, and robustness. Over the evaluated

scenarios, MetaCS-FL reduced total training time by up to 88.85% and total training energy by up to 84.45% relative to FedAvg (1), while typically reducing the number of rounds required to reach the target accuracy. These improvements stem from treating client selection as a multi-objective scheduling problem, rather than optimizing a single criterion, as in random selection (1), time- or energy-oriented scheduling (28), utility-driven selection (46), diversity-based selection (15), reputation-based selection (96), availability-budget selection (17), or availability-aware scheduling (22).

Under static client availability, MetaCS-FL achieved the strongest combined performance across the 100-client experiments. Its largest improvements occurred under CIFAR-10 non-IID, where it reduced total training time by 88.85% and energy consumption by 84.45% compared with FedAvg, while reducing the mean rounds to target accuracy from 92 to approximately 30. This behavior reflects a central challenge in non-IID FL: efficient clients are not necessarily representative, nor are representative clients necessarily efficient. MetaCS-FL addresses this by jointly considering makespan, energy, utility, diversity, and data-balancing criteria when generating the workload schedule.

The comparison with MEC and ECMTC highlights the limitations of optimizing system metrics independently of learning criteria. MEC minimizes makespan and therefore often selects many clients to reduce per-client workload and round duration. This can improve execution time, but it may increase energy consumption, selection overhead, and, in some cases, fail to produce sufficiently useful updates for reaching the target accuracy. ECMTC, in contrast, prioritizes energy consumption under a deadline, which tends to favor a smaller subset of energy-efficient clients. While this can reduce energy in some settings, it also concentrates participation, lowers fairness, and can bias the global model under non-IID data. Thus, time- and energy-aware scheduling alone is not sufficient when learning quality depends on data heterogeneity and client diversity.

The learning-oriented comparison methods show complementary limitations. Oort balances statistical utility and system efficiency through guided participant selection, which helps in several settings, but it does not explicitly optimize data balance, fairness, energy, and workload assignment together. DivFL selects diverse clients through a submodular diversity objective, but diversity alone does not guarantee low training time, low energy consumption, or fast convergence when selected clients have poor system characteristics. ECSM combines accuracy, reputation, and random selection, making it lightweight and occasionally competitive, but its dependence on historical information makes it less effective when clients are new, intermittent, or strongly heterogeneous. MetaCS-FL out-

performs these methods because it does not rely on a single learning signal. Instead, it uses utility and diversity as part of a broader scheduling objective.

The late-joining experiments further highlight the schedule-aware adaptation of MetaCS-FL. Across representative late-joining scenarios, it reduced total training time by up to 82.09% and total training energy by up to 76.79% relative to FedAvg. These reductions were observed whether late-arriving clients had favorable or poor system characteristics. When high-performance clients joined late, MetaCS-FL exploited them by assigning useful workload; when low-performance clients joined late, it limited their participation without excluding them. FedCAB and RIFLES are more directly designed for dynamic participation than Oort, DivFL, and ECSM, but their trade-offs differ: FedCAB improves fairness by compensating under-participating clients, whereas RIFLES filters clients using availability and expected execution feasibility. Both mechanisms can be useful, but neither consistently optimizes time, energy, convergence, and workload allocation together.

Under intermittent availability, MetaCS-FL again achieved the best overall performance across datasets and intermittent-availability settings, reducing total training time by up to 79.58% and total training energy by up to 62.84% relative to FedAvg. These reductions were not obtained by simply selecting fewer clients. Instead, MetaCS-FL uses past availability and completion outcomes to restrict the admissible workload capacities of less reliable clients. As a result, less reliable clients remain eligible for selection, but their assigned workload is controlled. This helps explain why MetaCS-FL can preserve broad participation while limiting the number of scheduled samples that fail to be processed after selection. Although Oort sometimes reports fewer failing clients or a lower failed-sample fraction, its time, energy, and round-to-accuracy performance are worse.

The privacy, scalability, and overhead analyses show that the proposed approach remains practical despite its broader optimization model. With differentially private class-distribution reports, MetaCS-FL preserved performance close to the non-private case, reducing training time by 0.65% and improving fairness by 8.88%, although at the cost of higher energy consumption and selection time. In the scalability experiments, MetaCS-FL incurred higher overhead than lightweight methods such as FedAvg, Oort, and ECSM, but remained substantially more scalable than exact scheduling approaches such as MEC and ECMTC. The bounded LNS runtime constraint, solution reuse, and triggered reselection criteria limit metaheuristic overhead while preserving the adaptability of MetaCS-FL. Here, selection-time values are cumulative across rounds and should not be interpreted as the latency or CPU utilization of one selection call.

The evidence should, nevertheless, be interpreted within the experimental scope: real FL executions use at most 100 clients, the separate scalability study simulates at most 1000 candidates and 40,000 tasks, devices and networks are emulated, and intermittent availability is generated rather than replayed from a production trace. Moreover, the sensitivity analysis varies one parameter at a time, while aggregate edge-resource utilization, instantaneous server CPU load, exhaustive weight interactions, and objective ablation are not measured. These boundaries do not change the reported comparisons, but they limit extrapolation to larger or operational deployments.

Overall, the experiments indicate that effective FL client selection requires balancing system efficiency, learning utility, data diversity, fairness, and reliability. Methods that optimize only one of these dimensions can perform well in specific cases, but their behavior becomes less stable under non-IID data, dynamic availability, or heterogeneous device capabilities. MetaCS-FL achieves more stable and consistent results because it integrates these dimensions into a single schedule-aware framework with adaptive workload assignment. This explains its large reductions in training time and energy, faster convergence in most settings, and robustness under both static and dynamic client availability.

7 Conclusion

Federated Learning (FL) enables collaborative model training across distributed clients without requiring the direct sharing of raw data (1, 3, 40). However, the practical deployment of FL, especially in cross-device scenarios, remains strongly affected by client heterogeneity, intermittent availability, limited energy resources, communication variability, and non-IID data distributions (14, 40, 16, 21). In this thesis, we addressed these challenges from the perspective of client selection, arguing that the selection of participating clients should not be treated only as a sampling or ranking problem, but as a scheduling problem in which the server jointly decides which clients participate and how much workload each selected client is assigned (7, 8, 99, 13).

The first step in this direction was the formulation of FL client selection as a task scheduling problem, in which clients are modeled as heterogeneous resources and the workload of a training or testing round is represented as a set of tasks, where each task denotes a configured quantity of local samples rather than statistically identical data (8, 60). This abstraction allows the server not only to decide which clients should participate, but also how much local data each selected client should process and, when class-aware scheduling is used, the estimated class composition of that assignment. Based on this formulation, two optimal scheduling algorithms were proposed: *MEC* and *ECMTC*. MEC minimizes the round makespan and then energy consumption, while ECMTC minimizes energy consumption and then makespan under a time constraint (28). Both algorithms rely on dynamic programming structures and provide optimal schedules for their respective objective orderings, showing that time and energy-aware client selection can be formally modeled and solved under well-defined assumptions.

Following the development of MEC and ECMTC, this thesis also proposed *MetaCS-FL*, a metaheuristic-based framework for client selection in synchronous cross-device FL systems (29). While MEC and ECMTC focus primarily on execution time and energy consumption, MetaCS-FL extends the scheduling perspective to a broader multi-objective setting that also considers model utility, class-distribution properties, and participation

diversity, while client reliability restricts each client's feasible workload capacity (46, 15, 17, 22). The server uses client profiles, availability information, noisy class-distribution statistics, historical performance, and reliability scores to search for schedules that balance system efficiency and learning-related objectives (27). In this work, MetaCS-FL uses the schedule generated by ECMTC as an initial solution and refines it through *Large Neighborhood Search* (LNS) (72). This design connects the optimal scheduling algorithms introduced earlier with a more flexible metaheuristic-based framework that can incorporate additional FL-specific objectives.

The main conceptual contribution of this thesis is therefore the formulation of FL client selection as a multi-objective task-to-resource scheduling problem (8, 60). This perspective differs from traditional client selection methods that primarily rank, sample, or filter clients according to utility, availability, fairness, or system metrics (7, 46, 15, 13). By explicitly controlling the number of tasks assigned to each client, the proposed approach provides a finer level of control over the participation of heterogeneous devices. This is particularly relevant in edge and mobile environments, where selecting a client and assigning it an excessive workload may be worse than assigning it a smaller, reliability-aware workload (99, 21). MetaCS-FL builds on this idea by allowing less reliable clients to remain eligible while restricting their assignment capacity, rather than excluding them outright. Through this mechanism, the framework aims to preserve participation opportunities while reducing the risk of stragglers, deadline violations, or incomplete processing.

The preliminary results show that scheduling-based client selection is effective for controlling time and energy costs in FL. MEC and ECMTC optimize time and energy with different priority orderings, improving makespan and energy consumption, respectively. However, because neither method accounts for learning-related criteria such as data distribution, client diversity, model utility, fairness, or reliability, their behavior can limit convergence and representativeness, especially under non-IID data (Section 4.3.4).

The broader evaluation shows that MetaCS-FL addresses these limitations through a multi-objective scheduling formulation that jointly considers system efficiency and learning quality. Under static availability, MetaCS-FL reduced total training time by up to 88.85% and total energy consumption by up to 84.45% compared with FedAvg, while reaching the target test accuracy with substantially fewer rounds, from 92 to approximately 30 on CIFAR-10 non-IID. In dynamic scenarios, it preserved this advantage, reducing time and energy by up to 82.09% and 76.79% under late-joining clients, and by up to 79.58% and 62.84% under intermittent availability, respectively. These results

confirm that adaptive workload assignment improves robustness under changing client availability while preserving faster convergence to the target accuracy (Section 6.4).

7.1 Limitations

The proposed approaches were developed and evaluated under specific assumptions. Accordingly, the following limitations clarify the scope of this work and indicate where broader validation or methodological extensions are still needed.

Synchronous and centralized FL scope. MetaCS-FL was designed and evaluated under a synchronous and centralized FL setting (1). In this configuration, a central server selects clients, distributes model parameters, waits for local processing to finish, and aggregates the received updates (1, 40). While this assumption is consistent with many practical FL systems, it also restricts the evaluation to scenarios where the round duration is affected by the slowest selected client (7, 8, 99). Consequently, MetaCS-FL was not designed for asynchronous, semi-asynchronous, hierarchical, decentralized, or peer-to-peer FL scenarios (102, 2). Extending the proposed scheduling abstraction to these settings would require revisiting the definition of a round, aggregation timing, stale updates, and client reliability (102, 21).

Fixed objective scalarization. Another limitation concerns the fixed scalarization strategy adopted by MetaCS-FL. In this work, the cost function combines multiple objectives through predefined weights, while additional parameters control the trade-offs between utility and class-distribution components. Although this makes the framework flexible and interpretable, it requires the user or system designer to choose suitable weights before execution, which is a common challenge in scalarized multi-objective optimization (106). The best trade-off may vary depending on the application, dataset, device population, network state, battery conditions, or training stage, as suggested by adaptive and stage-aware FL client selection and resource allocation strategies (81, 19, 88). For instance, early FL rounds may benefit from prioritizing class coverage and utility, while later rounds may emphasize fair participation and energy conservation, with reliability continuing to restrict feasible workload capacities. A fixed scalarization strategy may therefore limit the adaptability of the framework. Moreover, the one-factor-at-a-time sensitivity analysis does not quantify interactions among weights or establish whether all five weighted objectives are necessary, since an objective-ablation study was not performed.

Fixed local training configuration. The current framework assumes fixed local

training hyperparameters across clients. In the experiments, all clients use the same model architecture, optimizer, number of local epochs, batch size, and aggregation method, which isolates the impact of client selection and workload allocation. However, real cross-device FL systems may benefit from client-specific hyperparameter adaptation, since devices can differ in computational capability, energy availability, communication conditions, and data distributions (14, 40, 99). Such differences may require different local epochs, batch sizes, learning rates, compression levels, or update frequencies (88, 97). The current formulation controls how many tasks each client processes, but does not yet jointly optimize its local training configuration.

FedAvg-only aggregation. The aggregation strategy considered in this thesis is limited to FedAvg (1). Although FedAvg is a fundamental and widely used FL aggregation method (1, 40), other strategies may be more appropriate under strong data heterogeneity, partial participation, client drift, adversarial behavior, or asynchronous execution (14, 122, 123, 102). Since client selection and aggregation are strongly connected, MetaCS-FL may behave differently when combined with robust, personalized, clustered, adaptive, or staleness-aware aggregation methods.

Fixed workload size per round. The workload scheduled per round was fixed according to the experimental design. In this work, the server defines the total number of tasks for each round, while the scheduling mechanism decides how to distribute them among selected clients. However, in practical deployments, the workload itself could be dynamic, as FL systems may need to adapt client participation and resource allocation according to network conditions, device availability, and learning progress (10, 124, 88, 19). For example, the system could reduce the workload under poor network conditions, increase it when many reliable clients are available, or adjust it according to model convergence. Treating the total number of tasks as a decision variable would extend the scheduling problem from workload allocation to workload sizing.

LNS-centered metaheuristic configuration. MetaCS-FL was implemented and evaluated with an LNS-based metaheuristic configuration (72). Although this provides a controlled proof of concept, it does not fully explore the broader design space of metaheuristic search. Other metaheuristics, neighborhood structures, initialization strategies, acceptance criteria, hybrid approaches, and dynamic stopping rules may lead to different trade-offs between solution quality and selection overhead (114, 115, 117, 100). In particular, the number of iterations or allowed optimization time could be adapted according to historical solution quality, round urgency, client churn, or candidate-set changes.

Evaluated scalability and server-load scope. The empirical FL experiments use at most 100 clients, while the standalone scalability analysis covers at most 1,000 clients and 40,000 tasks. The reported selection time is cumulative across calls and should not be interpreted as per-call latency distributions, instantaneous or per-core CPU utilization, or contention with server services, which were not recorded. In addition, the LNS budget bounds the refinement phase but not the generation of the initial ECMTC schedule, which may dominate the overhead when the number of tasks becomes large.

Emulated evaluation environment. The evaluation environment relies on emulated devices and modeled network conditions rather than real edge or mobile devices. This enables controlled and reproducible experiments, following the broader role of simulation and benchmarking in FL research (48, 50). Nevertheless, real devices may introduce variability such as operating system scheduling, background applications, thermal throttling, battery degradation, wireless interference, mobility, and unpredictable disconnections (14, 40, 21). These factors may affect client profiles and scheduling decisions, making real-device deployment important for validating the proposed framework. Likewise, intermittent availability is generated by a Markov process rather than calibrated or replayed from production device-disconnection or preemptible-instance traces.

Simplified experimental task capacities. Although the framework supports client-specific capacity sets, the evaluation assumes that clients can process up to their full local dataset size in predefined task increments. This provides a structured assignment space, but may not reflect real user preferences or device constraints, such as battery level, charging state, data plan, privacy preferences, thermal state, or user activity (14, 40, 103). Future evaluations should consider more diverse and dynamic capacity sets, including irregular task granularities and client-defined maximum workloads. Larger task units could reduce scheduling cost at the expense of workload-allocation granularity, but this trade-off was not evaluated.

No speculative redundancy or aggregate edge-utilization metric. The current formulation assigns exactly the required round workload and does not reserve backup clients or duplicate tasks to mitigate failures that occur after scheduling. The evaluation also reports time, energy, accuracy, fairness, and selection overhead, but does not measure the aggregate utilization of the available edge resources.

Honest-client assumption. The current approach assumes honest clients and does not explicitly address adversarial behavior. Although the framework includes reliability-aware scheduling and privacy-preserving disclosure of noisy class histograms (27), it does

not model malicious clients that may falsify metadata, manipulate class distributions, poison model updates, perform backdoor attacks, or strategically influence selection decisions (125, 126, 127). Since MetaCS-FL relies on client profiles, historical metrics, and reported auxiliary information, adversarial robustness remains a limitation.

Dependence on recent profiling and historical information. MetaCS-FL depends on profiling and historical information to estimate client behavior. A short profiling stage is performed when clients enter the system, and the server updates its estimates with the latest available training or testing metrics whenever clients participate. However, these estimates may still become outdated when clients remain inactive for many rounds or when their network quality, battery state, device load, or user behavior changes between participations. In such cases, the scheduler may rely on outdated information and produce suboptimal decisions, especially in cross-device FL systems with heterogeneous resources and intermittent availability (14, 17, 18, 21). Profile refresh mechanisms can mitigate this problem, but they also introduce additional overhead. This work does not define optimal cadence or assess trade-offs between refresh cost and scheduling quality.

7.2 Future Work

The limitations discussed above point to opportunities to extend MetaCS-FL toward more adaptive, robust, and deployment-ready FL systems. The following directions outline possible extensions of the multi-objective scheduling perspective introduced in this thesis.

Asynchronous and semi-asynchronous FL. A natural extension is adapting MetaCS-FL to asynchronous and semi-asynchronous FL (102). In asynchronous FL, the server does not wait for all selected clients before updating the global model, reducing the impact of stragglers (7, 8, 99). However, this introduces challenges such as stale updates, client drift, and uneven contribution frequencies (102, 14, 40). Extending the scheduling formulation would require objectives related to update freshness, aggregation delay, staleness-aware utility, and continuous client availability (21). Under this extension, the scheduler could decide not only which clients process tasks, but also when their updates should be accepted, delayed, discounted, or discarded.

Decentralized and hierarchical coordination. Another research direction is extending MetaCS-FL beyond its current dependence on a permanent central server. Responsibilities currently assigned to the server, including collecting client profiles, maintaining reliability histories, constructing workload assignments, enforcing deadlines, and

aggregating model updates, could be distributed among hierarchical edge coordinators, an elected or rotating client coordinator, or clients participating through peer-to-peer consensus (2, 40). However, removing the physical central server would not eliminate the need for logical coordination. Clients would still need to agree on the round workload and task assignments, exchange the required scheduling information without exposing private data, prevent conflicting assignments, and tolerate coordinator or communication failures. Supporting these environments would require a distributed or consensus-based reformulation of the scheduling problem, together with an evaluation of its communication, privacy, fault-tolerance, and scalability trade-offs relative to the centralized approach.

Dynamic objective weighting. MetaCS-FL could adapt its objective weights during training instead of relying on fixed scalarization weights (106). For example, it could initially prioritize class coverage and utility, then increase the importance of energy efficiency and fairness as the model approaches convergence, following adaptive and stage-aware FL strategies (81, 19). The weights could also reflect application requirements, network state, deadline pressure, battery availability, or convergence behavior (88, 84). Reinforcement learning, Bayesian optimization, multi-armed bandits, or feedback-control mechanisms could guide this adaptation (79, 88, 84). Before deploying such adaptation, factorial experiments should quantify interactions among the weights, and objective ablations should test whether smaller subsets reproduce most of the end-to-end benefit.

Joint selection, workload, and hyperparameter optimization. Future work could extend MetaCS-FL beyond task assignment by jointly optimizing client selection, workload allocation, and local hyperparameters. In addition to assigning tasks, the server could configure local epochs, batch sizes, learning rates, compression settings, or early-stopping criteria, following broader joint client-selection and resource-allocation strategies for heterogeneous FL (10, 88, 19). This would make MetaCS-FL a more adaptive FL framework, especially useful when devices differ in capability or clients exhibit different learning behaviors under non-IID data (14, 120, 97).

Alternative aggregation methods. Future studies could evaluate MetaCS-FL with aggregation strategies beyond FedAvg, such as robust, personalized, clustered, adaptive, or staleness-aware aggregation (40, 14). This would clarify how client scheduling interacts with aggregation and whether different objectives should be emphasized depending on the aggregation mechanism (123, 102, 97). For example, robust aggregation may reduce unreliable or adversarial updates (128, 129), while personalized aggregation may change the importance of class-distribution coverage (97).

Adaptive workload sizing. The workload size could be made adaptive instead of fixed per round. The server could decide how many tasks to process based on system and learning conditions, following workload-aware scheduling and joint resource allocation in heterogeneous FL systems (124, 10, 88, 19). Under deadline pressure or low availability, the scheduler could reduce the workload, while increasing it when reliable and efficient clients are available. This is especially relevant for time-critical applications that must balance model improvement, latency, and energy constraints (89, 103). The task unit itself could also be adapted, coarsening assignments when scheduling overhead is critical and refining them when allocation precision is more valuable.

Speculative execution and deadline-aware result acceptance. A reliability-aware extension could assign tasks to backup clients and accept the first valid results before the deadline, while discarding late or duplicate copies. Such speculative execution would require duplicate-aware aggregation and an analysis of the added computation, communication, and energy costs relative to its latency and completion-rate benefits.

Expanded metaheuristic search. The metaheuristic component of MetaCS-FL could be replaced or extended beyond the current search procedure (72). Future work could evaluate genetic algorithms, simulated annealing, tabu search, ant colony optimization, particle swarm optimization, variable neighborhood search, or hybrid exact-metaheuristic methods, since different designs may lead to different trade-offs between solution quality and computational overhead (114, 115, 117, 100). The stopping criterion could also be made dynamic, allocating less search effort when solutions are stable and more effort under major availability changes or poor previous schedules. At larger scales, the exact ECMTC initialization could be replaced by a heuristic, approximate, or anytime initializer when its predicted runtime would exceed the selection budget, and the resulting quality–latency trade-off should be evaluated beyond 1,000 candidates.

Real-device deployment. A real-device deployment would strengthen the practical validation of MetaCS-FL by exposing it to constraints that are difficult to capture through emulation, such as battery dynamics, thermal throttling, wireless interference, mobility, and background system load (14, 40, 21). Such a deployment would enable the evaluation of overhead from profiling, privacy-preserving histogram disclosure, computing client selections, and coordinating tasks under realistic conditions. Production availability traces, including mobile disconnection logs or cloud preemptible-instance traces, could additionally calibrate reliability transitions and validate the resulting penalties against observed interruptions.

More realistic task-capacity evaluation. Although MetaCS-FL already supports client-specific capacity sets, future evaluations should use more realistic capacity definitions. In this work, clients were allowed to process tasks in fixed increments up to their total number of local samples (8, 60). Future experiments could instead define capacities according to battery level, charging state, user preferences, data plan, local storage, or application constraints, reflecting the heterogeneous and resource-constrained nature of cross-device FL (14, 40, 103, 21). This would include irregular task increments, client-defined limits, and dynamic availability of local resources. MEC, ECMTC, and MetaCS-FL could also enforce per-client remaining-battery or energy-budget constraints rather than optimizing only aggregate round energy.

Resource-utilization and server-load evaluation. Future experiments should report aggregate edge-resource utilization alongside the existing metrics and examine whether increased utilization translates into useful model progress. Server-side evaluation should separate per-call wall-clock latency from cumulative selection time, record CPU utilization over time and per core, and test contention with aggregation and other coordinator services.

Adversarial robustness. Adversarial robustness is another important research direction. Future work should investigate how MetaCS-FL behaves under data poisoning, model poisoning, backdoor attacks, unreliable metadata, falsified profiles, and strategic client behavior (125, 126, 127). This may require combining scheduling with trust management, anomaly detection, privacy-preserving aggregation, robust aggregation, reputation systems, and verifiable profiling mechanisms (45, 128, 129). Since the framework uses client-reported and historical information, protecting the selection process itself is an important step toward secure deployment (40).

Cell-free and next-generation wireless networks. The proposed framework could be extended to emerging network architectures, such as cell-free networks. In these environments, communication resources are distributed across multiple access points, so client selection may need to consider radio resource allocation, signal quality, mobility, interference, and cooperative transmission in addition to device and data properties (130, 131). Extending MetaCS-FL to cell-free or next-generation wireless networks would connect FL scheduling with communication-aware resource management (132, 133, 134, 10).

Time-critical FL applications. Finally, the proposed scheduling perspective can be explored in time-critical FL applications, such as intelligent transportation, healthcare monitoring, industrial control, disaster response, and real-time sensing systems, where

latency, resource constraints, and communication conditions are central concerns (14, 40, 89). In these scenarios, the value of a model update may depend not only on its accuracy contribution, but also on whether it arrives within a strict time budget (89, 10). Since MetaCS-FL already models time, energy, workload allocation, and deadlines, future work can extend it toward application-aware scheduling policies that account for urgency, quality-of-service requirements, and the cost of delayed updates. For millisecond-scale deadlines, this also requires measuring selection latency on the critical path and explicitly accepting bounded suboptimality or anytime decisions when exact initialization cannot finish within the application budget.

7.3 Final Remarks

This thesis introduced a scheduling-oriented view of client selection in Federated Learning. Starting from optimal time- and energy-aware algorithms and extending toward the MetaCS-FL framework, the proposed approach shows that client selection can be treated as a broader decision problem than simply choosing a subset of participants. By jointly considering participation and workload allocation, the proposed methods support more efficient and adaptive FL execution in cross-device systems.

Although the current work is limited to a synchronous centralized setting with fixed scalarization weights, fixed local hyperparameters, FedAvg aggregation, emulated devices, and honest clients, the multi-objective scheduling abstraction developed in this thesis opens several directions for future research. These include asynchronous execution, adaptive objective weighting, dynamic workload sizing, real-device deployment, adversarial robustness, communication-aware networking scenarios, and time-critical applications. Additional priorities are objective-interaction and ablation studies, trace-calibrated reliability validation, speculative deadline-aware execution, per-client energy budgets, and explicit edge- and server-utilization measurements at larger scales.

REFERENCES

1 MCMAHAN, Brendan *et al.* Communication-Efficient Learning of Deep Networks from Decentralized Data. *In:* PROCEEDINGS of the 20th International Conference on Artificial Intelligence and Statistics. Fort Lauderdale, FL, United States: PMLR, 2017. v. 54. (Proceedings of Machine Learning Research), p. 1273–1282. Available from: https://proceedings.mlr.press/v54/mcmahan17a.html.

2 SHEN, Sheng *et al.* From distributed machine learning to federated learning: In the view of data privacy and security. **Concurrency and Computation: Practice and Experience**, Wiley, v. 34, n. 16, e6002, 2022. DOI: 10.1002/cpe.6002.

3 YANG, Qiang *et al.* Federated Machine Learning: Concept and Applications. **ACM Transactions on Intelligent Systems and Technology**, ACM, v. 10, n. 2, 12:1–12:19, 2019. DOI: 10.1145/3298981.

4 RAJENDRAN, Suraj *et al.* Cloud-Based Federated Learning Implementation Across Medical Centers. **JCO Clinical Cancer Informatics**, American Society of Clinical Oncology, n. 5, p. 1–11, 2021. DOI: 10.1200/CCI.20.00060.

5 ABDULRAHMAN, Sawsan *et al.* FedMCCS: Multicriteria Client Selection Model for Optimal IoT Federated Learning. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 6, p. 4723–4735, 2021. DOI: 10.1109/JIOT.2020.3028742.

6 ALBELAIHI, Rana *et al.* Green Federated Learning via Energy-Aware Client Selection. *In:* GLOBECOM 2022 - 2022 IEEE Global Communications Conference. Rio de Janeiro, Brazil: IEEE, 2022. p. 13–18. DOI: 10.1109/GLOBECOM48099.2022.10001569.

7 NISHIO, Takayuki; YONETANI, Ryo. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. *In:* ICC 2019 - 2019 IEEE International Conference on Communications (ICC). Montreal, QC, Canada: IEEE, 2019. p. 1–7. DOI: 10.1109/ICC.2019.8761315.

8 PILLA, Laércio Lima. Optimal Task Assignment for Heterogeneous Federated Learning Devices. *In:* 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Portland, OR, United States: IEEE, 2021. p. 661–670. DOI: 10.1109/IPDPS49936.2021.00074.

9 WU, Hongda; WANG, Ping. Node Selection Toward Faster Convergence for Federated Learning on Non-IID Data. **IEEE Transactions on Network Science and Engineering**, IEEE, v. 9, n. 5, p. 3099–3111, 2022. DOI: 10.1109/TNSE.2022.3146399.

10 YU, Liangkun *et al.* Jointly Optimizing Client Selection and Resource Management in Wireless Federated Learning for Internet of Things. **IEEE Internet of Things Journal**, IEEE, v. 9, n. 6, p. 4385–4395, 2022. DOI: 10.1109/JIOT.2021.3103715.

11 ZHANG, Ruilin; XU, Zhenan; YIN, Hao. Scout: An Efficient Federated Learning Client Selection Algorithm Driven by Heterogeneous Data and Resource. *In:* 2023 IEEE International Conference on Joint Cloud Computing (JCC). Athens, Greece: IEEE, 2023. p. 46–49. DOI: 10.1109/JCC59055.2023.00012.

12 ZHENG, Jingjing *et al.* Federated Learning for Energy-balanced Client Selection in Mobile Edge Computing. *In:* 2021 International Wireless Communications and Mobile Computing (IWCMC). Harbin City, China: IEEE, 2021. p. 1942–1947. DOI: 10.1109/IWCMC51323.2021.9498853.

13 LI, Jian; CHEN, Tongbao; TENG, Shaohua. A comprehensive survey on client selection strategies in federated learning. **Computer Networks**, Elsevier, v. 251, p. 110663, 2024. DOI: 10.1016/j.comnet.2024.110663.

14 LI, Tian *et al.* Federated Learning: Challenges, Methods, and Future Directions. **IEEE Signal Processing Magazine**, IEEE, v. 37, n. 3, p. 50–60, 2020. DOI: 10.1109/MSP.2020.2975749.

15 BALAKRISHNAN, Ravikumar *et al.* Diverse Client Selection for Federated Learning via Submodular Maximization. *In:* INTERNATIONAL Conference on Learning Representations. Virtual Conference: ICLR, 2022. Available from: https://openreview.net/forum?id=nwKXyFvaUm.

16 FU, Lei *et al.* Client Selection in Federated Learning: Principles, Challenges, and Opportunities. **IEEE Internet of Things Journal**, IEEE, Piscataway, NJ, United States, v. 10, n. 24, p. 21811–21819, 2023. DOI: 10.1109/JIOT.2023.3299573.

17 BAO, Yunkai *et al.* Federated Learning with Client Availability Budgets. *In:* GLOBECOM 2023 - 2023 IEEE Global Communications Conference. Kuala Lumpur, Malaysia: IEEE, 2023. p. 1902–1907. DOI: 10.1109/GLOBECOM54140.2023.10437111.

18 RODIO, Angelo *et al.* Federated Learning under Heterogeneous and Correlated Client Availability. *In:* IEEE INFOCOM 2023 - IEEE Conference on Computer

Communications. Piscataway, NJ, United States: IEEE, 2023. p. 1–10. DOI: 10.1109/INFOCOM53939.2023.10228876.

19 WU, Minghong *et al.* Towards Stagewise and Energy-Accuracy-balanced Client Selection and Resource Allocation over Dynamic Federated Learning Networks. *In:* 2024 IEEE/CIC International Conference on Communications in China (ICCC). Chengdu, China: IEEE, 2024. p. 839–844. DOI: 10.1109/ICCC62479.2024.10681991.

20 XIANG, Ming *et al.* Efficient Federated Learning against Heterogeneous and Non-stationary Client Unavailability. *In:* ADVANCES in Neural Information Processing Systems. Vancouver, Canada: Neural Information Processing Systems Foundation, Inc., 2024. v. 37, p. 104281–104328. DOI: 10.52202/079017-3313.

21 RIBERO, Mónica; VIKALO, Haris; VECIANA, Gustavo de. Federated Learning at Scale: Addressing Client Intermittency and Resource Constraints. **IEEE Journal of Selected Topics in Signal Processing**, IEEE, Piscataway, NJ, United States, v. 19, n. 1, p. 60–73, 2025. DOI: 10.1109/JSTSP.2024.3430118.

22 ALOSAIME, Sara; JHUMKA, Arshad. RIFLES: Resource-effIcient Federated LEarning via Scheduling. *In:* 2025 IEEE 22nd International Conference on Mobile Ad-Hoc and Smart Systems (MASS). Piscataway, NJ, United States: IEEE, 2025. p. 160–166. DOI: 10.1109/MASS66014.2025.00034.

23 XIE, Cong; KOYEJO, Sanmi; GUPTA, Indranil. Asynchronous Federated Optimization. **arXiv preprint arXiv:1903.03934**, arXiv, Ithaca, NY, United States, 2019. arXiv: 1903.03934 [cs.LG].

24 NGUYEN, John *et al.* Federated Learning with Buffered Asynchronous Aggregation. *In:* PROCEEDINGS of The 25th International Conference on Artificial Intelligence and Statistics. Virtual Conference: PMLR, 2022. v. 151, p. 3581–3607. Available from: https://proceedings.mlr.press/v151/nguyen22b.html.

25 KRIZHEVSKY, Alex. **Learning Multiple Layers of Features from Tiny Images**. Toronto, Canada, 2009. Accessed: 2024-01-01. Available from: https://www.cs.toronto.edu/~kriz/cifar.html.

26 SARAVIA, Elvis *et al.* CARER: Contextualized Affect Representations for Emotion Recognition. *In:* PROCEEDINGS of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 3687–3697. DOI: 10.18653/v1/D18-1404.

27 WEI, Kang *et al.* Federated Learning With Differential Privacy: Algorithms and Performance Analysis. **IEEE Transactions on Information Forensics and**

Security, IEEE, Piscataway, NJ, United States, v. 15, p. 3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575.

28 NUNES, Alan L. *et al.* Optimal Time and Energy-Aware Client Selection Algorithms for Federated Learning on Heterogeneous Resources. *In:* 2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Hilo, Hawaii, United States: IEEE, 2024. p. 148–158. DOI: 10.1109/SBAC-PAD63648.2024.00021.

29 NUNES, Alan L. *et al.* MetaCS-FL: A Metaheuristic-Based Framework for Client Selection in Federated Learning Systems. **Future Generation Computer Systems**, v. 185, p. 108707, 2026. ISSN 0167-739X. DOI: 10.1016/j.future.2026.108707.

30 BISHOP, Christopher M. **Pattern Recognition and Machine Learning**. New York, NY, United States: Springer, 2006. ISBN 978-0-387-31073-2.

31 HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. 2. ed. New York, NY, United States: Springer, 2009. ISBN 978-0-387-84857-0.

32 GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. Cambridge, MA, United States: MIT Press, 2016. ISBN 978-0-262-03561-3. Available from: https://www.deeplearningbook.org.

33 ROBBINS, Herbert; MONRO, Sutton. A Stochastic Approximation Method. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 22, n. 3, p. 400–407, 1951. DOI: 10.1214/aoms/1177729586.

34 BOTTOU, Léon. Large-Scale Machine Learning with Stochastic Gradient Descent. *In:* PROCEEDINGS of COMPSTAT 2010: 19th International Conference on Computational Statistics. Heidelberg, Germany: Springer, 2010. p. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

35 KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. *In:* INTERNATIONAL Conference on Learning Representations. San Diego, CA, United States: ICLR, 2015. arXiv: 1412.6980 [cs.LG].

36 LECUN, Yann *et al.* Gradient-based Learning Applied to Document Recognition. **Proceedings of the IEEE**, IEEE, Piscataway, NJ, United States, v. 86, n. 11, p. 2278–2324, 1998. DOI: 10.1109/5.726791.

37 HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, MIT Press, Cambridge, MA, United States, v. 9, n. 8, p. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

38 BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. *In:* INTERNATIONAL Conference on Learning Representations. San Diego, CA, United States: ICLR, 2015. arXiv: 1409.0473 [cs.CL].

39 VASWANI, Ashish *et al.* Attention Is All You Need. *In:* ADVANCES in Neural Information Processing Systems. Long Beach, CA, United States: Curran Associates, Inc., 2017. v. 30. Available from: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

40 KAIROUZ, Peter *et al.* Advances and Open Problems in Federated Learning. **Foundations and Trends in Machine Learning**, Now Publishers Inc., Hanover, MA, United States, v. 14, n. 1–2, p. 1–210, 2021. DOI: 10.1561/2200000083.

41 LI, Tian *et al.* Federated Optimization in Heterogeneous Networks. *In:* PROCEEDINGS of Machine Learning and Systems. Austin, TX, United States: MLSys, 2020. v. 2, p. 429–450. Available from: https://proceedings.mlsys.org/paper/2020/file/1f5fe83998a09396ebe6477d9475ba0c-Paper.pdf.

42 CHAI, Zheng *et al.* TiFL: A Tier-Based Federated Learning System. *In:* PROCEEDINGS of the 29th International Symposium on High-Performance Parallel and Distributed Computing. Stockholm, Sweden: ACM, 2020. p. 125–136. DOI: 10.1145/3369583.3392686.

43 DWORK, Cynthia *et al.* Calibrating Noise to Sensitivity in Private Data Analysis. *In:* THEORY of Cryptography: Third Theory of Cryptography Conference, TCC 2006, Proceedings. Berlin, Germany: Springer, 2006. v. 3876. (Lecture Notes in Computer Science), p. 265–284. DOI: 10.1007/11681878_14.

44 DWORK, Cynthia; ROTH, Aaron. The Algorithmic Foundations of Differential Privacy. **Foundations and Trends in Theoretical Computer Science**, Now Publishers Inc., Hanover, MA, United States, v. 9, n. 3–4, p. 211–407, 2014. DOI: 10.1561/0400000042.

45 BONAWITZ, Keith *et al.* Practical Secure Aggregation for Privacy-Preserving Machine Learning. *In:* PROCEEDINGS of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, TX, United States: ACM, 2017. p. 1175–1191. DOI: 10.1145/3133956.3133982.

46 LAI, Fan *et al.* Oort: Efficient Federated Learning via Guided Participant Selection. *In:* 15TH USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). Berkeley, CA, United States: USENIX Association, 2021. p. 19–35. ISBN 978-1-939133-22-9. Available from: https://www.usenix.org/conference/osdi21/presentation/lai.

47 TENSORFLOW. **TensorFlow Federated**. Mountain View, CA, United States: Google, 2026. Accessed: 2026-05-01. Available from: https://www.tensorflow.org/federated.

48 HE, Chaoyang *et al.* FedML: A Research Library and Benchmark for Federated Machine Learning. **arXiv preprint arXiv:2007.13518**, arXiv, Ithaca, NY, United States, 2020. arXiv: 2007.13518 [cs.LG].

49 NVIDIA. **NVIDIA FLARE**. Santa Clara, CA, United States: NVIDIA, 2026. Accessed: 2026-05-01. Available from: https://nvflare.readthedocs.io/.

50 LAI, Fan *et al.* FedScale: Benchmarking Model and System Performance of Federated Learning at Scale. *In:* PROCEEDINGS of the 39th International Conference on Machine Learning. Baltimore, MD, United States: PMLR, 2022. v. 162. (Proceedings of Machine Learning Research), p. 11814–11827. Available from: https://proceedings.mlr.press/v162/lai22a.html.

51 POLATO, Mirko. Fluke: Federated Learning Utility Framework for Experimentation and Research. **Future Generation Computer Systems**, Elsevier, Amsterdam, Netherlands, v. 177, p. 108241, 2026. DOI: 10.1016/j.future.2025.108241.

52 CANTALI, Gökcan; GÜR, Gürkan; STILLER, Burkhard. FedSynthesis: A Flower-Based Framework for Carbon-Reduced Federated Learning. *In:* PROCEEDINGS of the 18th IEEE/ACM International Conference on Utility and Cloud Computing. New York, NY, United States: Association for Computing Machinery, 2025. (UCC '25), p. 10–19. DOI: 10.1145/3773274.3774265.

53 BEUTEL, Daniel J. *et al.* Flower: A Friendly Federated Learning Research Framework. **arXiv preprint arXiv:2007.14390**, arXiv, Ithaca, NY, United States, 2020. arXiv: 2007.14390 [cs.DC].

54 JAIN, Raj. Fairness Measures. *In:* THE ART OF COMPUTER SYSTEMS PERFORMANCE ANALYSIS: TECHNIQUES FOR EXPERIMENTAL DESIGN, MEASUREMENT, SIMULATION, AND MODELING. New York, NY, USA: John Wiley & Sons, 1991. chap. 35, p. 577–583. ISBN 978-0-471-50336-1.

55 PINEDO, Michael L. **Scheduling: Theory, Algorithms, and Systems**. 6. ed. Cham: Springer Cham, 2022. ISBN 978-3-031-05921-6. DOI: 10.1007/978-3-031-05921-6.

56 BRUCKER, Peter. **Scheduling Algorithms**. 5. ed. Berlin, Heidelberg: Springer Berlin, Heidelberg, 2007. ISBN 978-3-540-69516-5. DOI: 10.1007/978-3-540-69516-5.

57 PILLA, Laércio Lima. **Scheduling Everything, Everywhere, All at Once**. Bordeaux, France: Université de Bordeaux, 2026. Distributed, Parallel, and Cluster Computing [cs.DC]. Available from: https://hal.science/tel-05624925.

58 GRAHAM, Ronald L. *et al.* Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. **Annals of Discrete Mathematics**, Elsevier, v. 5, p. 287–326, 1979. DOI: 10.1016/S0167-5060(08)70356-X.

59 GAREY, Michael R.; JOHNSON, David S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY, United States: W. H. Freeman & Co., 1990. ISBN 978-0-7167-1045-5. DOI: 10.5555/574848.

60 PILLA, Laércio Lima. Scheduling Algorithms for Federated Learning With Minimal Energy Consumption. **IEEE Transactions on Parallel and Distributed Systems**, v. 34, n. 4, p. 1215–1226, 2023. DOI: 10.1109/TPDS.2023.3240833.

61 BELLMAN, Richard. **Dynamic Programming**. Mineola, NY, United States: Dover Publications, 2003. ISBN 978-0-486-42809-3.

62 CORMEN, Thomas H. *et al.* **Introduction to Algorithms**. 3. ed. Cambridge, MA: The MIT Press, 2009. ISBN 978-0-262-03384-8.

63 KELLERER, Hans; PFERSCHY, Ulrich; PISINGER, David. **Knapsack Problems**. Berlin, Heidelberg: Springer, 2004. ISBN 978-3-540-24777-7. DOI: 10.1007/978-3-540-24777-7.

64 GLOVER, Fred; KOCHENBERGER, Gary A. (eds.). **Handbook of Metaheuristics**. New York, NY: Springer New York, NY, 2003. v. 57. (International Series in Operations Research & Management Science). ISBN 978-1-4020-7263-5. DOI: 10.1007/b101874.

65 TALBI, El-Ghazali. **Metaheuristics: From Design to Implementation**. Hoboken, NJ: John Wiley & Sons, 2009. ISBN 978-0-470-27858-1. DOI: 10.1002/9780470496916.

66 BLUM, Christian; ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. **ACM Computing Surveys**, ACM, v. 35, n. 3, p. 268–308, 2003. DOI: 10.1145/937503.937505.

67 KIRKPATRICK, Scott; GELATT, C. Daniel; VECCHI, Mario P. Optimization by Simulated Annealing. **Science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. DOI: 10.1126/science.220.4598.671.

68 GLOVER, Fred; LAGUNA, Manuel. **Tabu Search**. New York, NY: Springer, 1997. ISBN 978-0-7923-9965-0. DOI: 10.1007/978-1-4615-6089-0.

69 LOURENÇO, Helena Ramalhinho; MARTIN, Olivier C.; STÜTZLE, Thomas. Iterated Local Search: Framework and Applications. *In:* HANDBOOK OF METAHEURISTICS. 3. ed. Cham: Springer, 2019. v. 272. (International Series in Operations Research & Management Science). p. 129–168. ISBN 978-3-319-91086-4. DOI: 10.1007/978-3-319-91086-4_5.

70 HANSEN, Pierre; MLADENOVIĆ, Nenad; PÉREZ, José A. Moreno. Variable Neighbourhood Search: Methods and Applications. **Annals of Operations Research**, Springer, v. 175, n. 1, p. 367–407, 2010. DOI: 10.1007/s10479-009-0657-6.

71 SHAW, Paul. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *In:* PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING – CP98. Berlin, Heidelberg: Springer, 1998. v. 1520. (Lecture Notes in Computer Science), p. 417–431. ISBN 978-3-540-49481-2. DOI: 10.1007/3-540-49481-2_30.

72 PISINGER, David; RØPKE, Stefan. Large Neighborhood Search. *In:* HANDBOOK OF METAHEURISTICS. 3. ed. Cham: Springer, 2019. v. 272. (International Series in Operations Research & Management Science). p. 99–127. ISBN 978-3-319-91086-4. DOI: 10.1007/978-3-319-91086-4_4.

73 HOLLAND, John H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**. Cambridge, MA: The MIT Press, 1992. ISBN 978-0-262-58111-0. DOI: 10.7551/mitpress/1090.001.0001.

74 KENNEDY, James; EBERHART, Russell C. Particle Swarm Optimization. *In:* PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS. Perth, Australia: IEEE, 1995. v. 4, p. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

75 DORIGO, Marco; BIRATTARI, Mauro; STÜTZLE, Thomas. Ant Colony Optimization. **IEEE Computational Intelligence Magazine**, IEEE, v. 1, n. 4, p. 28–39, 2006. DOI: 10.1109/MCI.2006.329691.

76 TAHIR, Mehreen; ALI, Muhammad Intizar. Client Selection in Federated Learning: Challenges, Strategies, and Contextual Considerations. *In:* FEDERATED LEARNING SYSTEMS. Cham: Springer, 2025. v. 832. (Studies in Computational Intelligence). p. 43–62. ISBN 978-3-031-78841-3. DOI: 10.1007/978-3-031-78841-3_3.

77 MA, Xiaodong *et al.* A State-of-the-Art Survey on Solving Non-IID Data in Federated Learning. **Future Generation Computer Systems**, Elsevier, v. 135, p. 244–258, 2022. DOI: 10.1016/j.future.2022.05.003.

78 QI, Tao *et al.* FedSampling: A Better Sampling Strategy for Federated Learning. *In:* PROCEEDINGS OF THE THIRTY-SECOND INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI-23. Macao, SAR: International Joint Conferences on Artificial Intelligence, 2023. p. 4154–4162. ISBN 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/462.

79 WANG, Hao *et al.* Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. *In:* INFOCOM 2020 - IEEE CONFERENCE ON COMPUTER COMMUNICATIONS. Toronto, Canada: IEEE, 2020. p. 1698–1707. DOI: 10.1109/INFOCOM41043.2020.9155494.

80 DENG, Yongheng *et al.* AUCTION: Automated and Quality-Aware Client Selection Framework for Efficient Federated Learning. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 33, n. 8, p. 1996–2009, 2022. DOI: 10.1109/TPDS.2021.3134647.

81 LI, Qingming *et al.* AdaFL: Adaptive Client Selection and Dynamic Contribution Evaluation for Efficient Federated Learning. *In:* ICASSP 2024 - 2024 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING. Seoul, Korea: IEEE, 2024. p. 6645–6649. DOI: 10.1109/ICASSP48485.2024.10447356.

82 YAN, Gang *et al.* CriticalFL: A Critical Learning Periods Augmented Client Selection Framework for Efficient Federated Learning. *In:* PROCEEDINGS OF THE 29TH ACM SIGKDD CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. Long Beach, CA, USA: ACM, 2023. p. 2898–2907. ISBN 979-8-4007-0103-0. DOI: 10.1145/3580305.3599293.

83 OUYANG, Jinhao; LIU, Yuan. Learning Efficiency Maximization for Wireless Federated Learning With Heterogeneous Data and Clients. **IEEE Transactions**

on Cognitive Communications and Networking, IEEE, v. 10, n. 6, p. 2282–2295, 2024. DOI: 10.1109/TCCN.2024.3394889.

84 ZHU, Konglin *et al.* Client Selection for Federated Learning Using Combinatorial Multi-Armed Bandit Under Long-Term Energy Constraint. **Computer Networks**, Elsevier, v. 250, p. 110512, 2024. DOI: 10.1016/j.comnet.2024.110512.

85 MACIEL, Filipe *et al.* Federated Learning Energy Saving Through Client Selection. **Pervasive and Mobile Computing**, Elsevier, v. 103, p. 101948, 2024. DOI: 10.1016/j.pmcj.2024.101948.

86 YANG, Hailiang; WANG, Zhongkun; CUI, Laizhong. Energy-Efficient Federated Learning in Mobile Edge Computing via Fine-Grained Energy Planning Client Selection. **IEEE Transactions on Mobile Computing**, IEEE, p. 1–14, 2026. Early Access. DOI: 10.1109/TMC.2026.3668371.

87 MISHRA, Rahul *et al.* Fed-RAC: Resource-Aware Clustering for Tackling Heterogeneity of Participants in Federated Learning. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 35, n. 7, p. 1207–1220, 2024. DOI: 10.1109/TPDS.2024.3379933.

88 ZHENG, Feng; SUN, Yuze; NI, Bin. FedAEB: Deep Reinforcement Learning Based Joint Client Selection and Resource Allocation Strategy for Heterogeneous Federated Learning. **IEEE Transactions on Vehicular Technology**, IEEE, v. 73, n. 6, p. 8835–8846, 2024. DOI: 10.1109/TVT.2024.3359860.

89 GAO, Weifeng *et al.* Resource Allocation for Latency-Aware Federated Learning in Industrial Internet of Things. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 12, p. 8505–8513, 2021. DOI: 10.1109/TII.2021.3073642.

90 SHI, Fang *et al.* The Analysis and Optimization of Volatile Clients in Over-the-Air Federated Learning. **IEEE Transactions on Mobile Computing**, IEEE, v. 23, n. 12, p. 13144–13157, 2024. DOI: 10.1109/TMC.2024.3427709.

91 SHI, Fang *et al.* Dynamic Client Selection for Over-the-Air Federated Learning Networks. **IEEE Internet of Things Journal**, IEEE, v. 12, n. 12, p. 18508–18519, 2025. DOI: 10.1109/JIOT.2025.10855420.

92 BANERJEE, Roopkatha *et al.* Federated Learning Within Global Energy Budget over Heterogeneous Edge Accelerators. *In:* EURO-PAR 2025: PARALLEL PROCESSING. Cham: Springer, 2026. v. 15900. (Lecture Notes in Computer Science). p. 264–278. ISBN 978-3-031-99854-6. DOI: 10.1007/978-3-031-99854-6_18.

93 LIU, Ji *et al.* AEDFL: Efficient Asynchronous Decentralized Federated Learning with Heterogeneous Devices. *In:* PROCEEDINGS OF THE 2024 SIAM INTERNATIONAL CONFERENCE ON DATA MINING (SDM). Houston, TX, USA: SIAM, 2024. p. 833–841. ISBN 978-1-61197-803-2. DOI: 10.1137/1.9781611978032.95.

94 BENARBA, Nawel; BOUCHENAK, Sara. Bias in Federated Learning: A Comprehensive Survey. **ACM Computing Surveys**, Association for Computing Machinery, New York, NY, USA, v. 57, n. 11, 291:1–291:36, 2025. DOI: 10.1145/3735125.

95 LI, Tian *et al.* Fair Resource Allocation in Federated Learning. *In:* INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS. Addis Ababa, Ethiopia: OpenReview.net, 2020. arXiv: 1905.10497 [cs.LG].

96 PUTRA, Made Adi Paramartha *et al.* ECSM: An Ensembled Client Selection Mechanism for Efficient Federated Learning. *In:* 2023 IEEE INTERNATIONAL CONFERENCE ON INDUSTRY 4.0, ARTIFICIAL INTELLIGENCE, AND COMMUNICATIONS TECHNOLOGY (IAICT). Pattaya, Thailand: IEEE, 2023. p. 13–17. DOI: 10.1109/IAICT59002.2023.10205864.

97 SINGH, Neha; ADHIKARI, Mainak. FedTune-SGM: A Stackelberg-Driven Personalized Federated Learning Strategy for Edge Networks. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 36, n. 4, p. 791–802, 2025. DOI: 10.1109/TPDS.2025.3543368.

98 CASTILLO, Edwin F.; CAICEDO, Oscar M.; FONSECA, Nelson L. S. da. Effective, Explainable, and Trustworthy Client Selection in Federated Learning. **IEEE Open Journal of the Computer Society**, IEEE, v. 7, p. 769–781, 2026. DOI: 10.1109/OJCS.2026.3683012.

99 WANG, Cong; YANG, Yuanyuan; ZHOU, Pengzhan. Towards Efficient Scheduling of Federated Mobile Devices Under Computational and Statistical Heterogeneity. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 32, n. 2, p. 394–410, 2021. DOI: 10.1109/TPDS.2020.3023905.

100 DRISS, Maryam Ben *et al.* GWO-Boosted Multi-Attribute Client Selection for Over-The-Air Federated Learning. *In:* 2024 20TH INTERNATIONAL CONFERENCE ON THE DESIGN OF RELIABLE COMMUNICATION NETWORKS (DRCN). Nice, France: IEEE, 2024. p. 62–69. DOI: 10.1109/DRCN60692.2024.10539149.

101 CHAHOUD, Mario *et al.* On-Demand-FL: A Dynamic and Efficient Multicriteria Federated Learning Client Deployment Scheme. **IEEE Internet of Things**

Journal, IEEE, v. 10, n. 18, p. 15822–15834, 2023. DOI: 10.1109/JIOT.2023.3265564.

102 SPRAGUE, Michael R. *et al.* Asynchronous Federated Learning for Geospatial Applications. *In:* ECML PKDD 2018 WORKSHOPS. Cham: Springer, 2019. v. 967. (Communications in Computer and Information Science). p. 21–28. ISBN 978-3-030-14880-5. DOI: 10.1007/978-3-030-14880-5_2.

103 LI, Yuchen *et al.* Energy-Aware, Device-to-Device Assisted Federated Learning in Edge Computing. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 34, n. 7, p. 2138–2154, 2023. DOI: 10.1109/TPDS.2023.3277423.

104 ALBASEER, Abdullatif *et al.* Novel Approach for Curbing Unfair Energy Consumption and Biased Model in Federated Edge Learning. **IEEE Transactions on Green Communications and Networking**, v. 8, n. 2, p. 865–877, 2024. DOI: 10.1109/TGCN.2024.3350735.

105 QIU, Xinchi *et al.* A First Look into the Carbon Footprint of Federated Learning. **Journal of Machine Learning Research**, v. 24, n. 129, p. 1–23, 2023. Available from: https://jmlr.org/papers/v24/21-0445.html.

106 ANDERSSON, Johan. **A Survey of Multiobjective Optimization in Engineering Design**. Linköping, Sweden, 2000.

107 BOLZE, Raphaël *et al.* Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. **The International Journal of High Performance Computing Applications**, SAGE Publications, v. 20, n. 4, p. 481–494, 2006. ISSN 1094-3420. DOI: 10.1177/1094342006070078. Available from: https://hal.inria.fr/hal-00684943.

108 SANDLER, Mark *et al.* MobileNetV2: Inverted Residuals and Linear Bottlenecks. *In:* PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT, United States: IEEE, 2018. p. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

109 NOUREDDINE, Adel. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. *In:* 2022 18th International Conference on Intelligent Environments (IE). Barcelona, Spain: IEEE, 2022. p. 1–4. DOI: 10.1109/IE54923.2022.9826760. Available from: https://hal.science/hal-03608223.

110 NEAR, Joseph P. *et al.* **Guidelines for Evaluating Differential Privacy Guarantees**. Gaithersburg, MD, 2025. DOI: 10.6028/NIST.SP.800-226.

111 MAGURRAN, Anne E.; MCGILL, Brian J. Diversity Indices. *In:* BIOLOGICAL DIVERSITY: FRONTIERS IN MEASUREMENT AND ASSESSMENT. Oxford, UK: Oxford University Press, 2011. chap. 2, p. 38–76. ISBN 978-0-19-958067-5.

112 MIETTINEN, Kaisa. **Nonlinear Multiobjective Optimization**. New York, NY, United States: Springer, 1998. v. 12. (International Series in Operations Research & Management Science). ISBN 978-0-7923-8278-2. DOI: 10.1007/978-1-4615-5563-6.

113 STANLEY, Richard P. **Enumerative Combinatorics**. 2. ed. Cambridge: Cambridge University Press, 2011. v. 1. (Cambridge Studies in Advanced Mathematics). ISBN 978-1-107-60262-5. DOI: 10.1017/CBO9781139058520.

114 ROPKE, Stefan; PISINGER, David. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. **Transportation Science**, v. 40, n. 4, p. 455–472, 2006. DOI: 10.1287/trsc.1050.0135.

115 PISINGER, David; ROPKE, Stefan. A General Heuristic for Vehicle Routing Problems. **Computers & Operations Research**, v. 34, n. 8, p. 2403–2435, 2007. DOI: 10.1016/j.cor.2005.09.012.

116 CORDEAU, Jean-François *et al.* Scheduling Technicians and Tasks in a Telecommunications Company. **Journal of Scheduling**, v. 13, n. 4, p. 393–409, 2010. DOI: 10.1007/s10951-010-0188-7.

117 MULLER, Laurent Flindt. An Adaptive Large Neighborhood Search Algorithm for the Resource-Constrained Project Scheduling Problem. *In:* MIC 2009: The VIII Metaheuristics International Conference. Hamburg, Germany: University of Hamburg, 2009. Available from: https://backend.orbit.dtu.dk/ws/files/4002776/mic09-152-Muller_b.pdf.

118 XIAO, Han; RASUL, Kashif; VOLLGRAF, Roland. **Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms**. Ithaca, NY: arXiv, 2017. arXiv: 1708.07747 [cs.LG].

119 GO, Alec; BHAYANI, Richa; HUANG, Lei. **Twitter Sentiment Classification using Distant Supervision**. Stanford, CA, United States, 2009. p. 1–12. Available from: https://www-cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf.

120 LI, Qinbin *et al.* Federated Learning on Non-IID Data Silos: An Experimental Study. *In:* 2022 IEEE 38th International Conference on Data Engineering (ICDE). Kuala Lumpur, Malaysia: IEEE, 2022. p. 965–978. DOI: 10.1109/ICDE53745.2022.00077.

121 YAN, Jiangting; PU, Pengju; JIANG, Liheng. Emotion-RGC net: A novel approach for emotion recognition in social media using RoBERTa and Graph Neural Networks. **PLOS ONE**, v. 20, n. 3, e0318524, 2025. DOI: 10.1371/journal.pone.0318524.

122 LI, Xiang *et al.* On the Convergence of FedAvg on Non-IID Data. *In:* INTERNATIONAL Conference on Learning Representations. Virtual Conference: OpenReview.net, 2020. Available from: https://openreview.net/forum?id=HJxNAnVtDS.

123 KARIMIREDDY, Sai Praneeth *et al.* SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *In:* PROCEEDINGS of the 37th International Conference on Machine Learning. Virtual Event: PMLR, 2020. v. 119. (Proceedings of Machine Learning Research), p. 5132–5143. Available from: https://proceedings.mlr.press/v119/karimireddy20a.html.

124 LI, Li *et al.* Federated Learning with Workload-Aware Client Scheduling in Heterogeneous Systems. **Neural Networks**, v. 154, p. 560–573, 2022. ISSN 0893-6080. DOI: 10.1016/j.neunet.2022.07.030.

125 BHAGOJI, Arjun Nitin *et al.* Analyzing Federated Learning through an Adversarial Lens. *In:* PROCEEDINGS of the 36th International Conference on Machine Learning. Long Beach, CA, United States: PMLR, 2019. v. 97. (Proceedings of Machine Learning Research), p. 634–643. Available from: https://proceedings.mlr.press/v97/bhagoji19a.html.

126 BAGDASARYAN, Eugene *et al.* How To Backdoor Federated Learning. *In:* PROCEEDINGS of the Twenty Third International Conference on Artificial Intelligence and Statistics. Online: PMLR, 2020. v. 108. (Proceedings of Machine Learning Research), p. 2938–2948. Available from: https://proceedings.mlr.press/v108/bagdasaryan20a.html.

127 LYU, Lingjuan; YU, Han; YANG, Qiang. **Threats to Federated Learning: A Survey**. Ithaca, NY, United States: arXiv, 2020. arXiv: 2003.02133 [cs.CR].

128 BLANCHARD, Peva *et al.* Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. *In:* ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. Long Beach, CA, United States: Curran Associates, Inc., 2017. v. 30, p. 119–129. Available from: https://papers.nips.cc/paper/6617-machine-learning-with-adversaries-byzantine-tolerant-gradient-descent.

129 YIN, Dong *et al.* Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. *In:* PROCEEDINGS of the 35th International Conference on Machine Learning. Stockholm, Sweden: PMLR, 2018. v. 80, p. 5650–5659. Available from: https://proceedings.mlr.press/v80/yin18a.html.

130 CHEN, Sheng *et al.* A Survey on User-Centric Cell-Free Massive MIMO Systems. **Digital Communications and Networks**, v. 8, n. 5, p. 695–719, 2022. DOI: 10.1016/j.dcan.2021.12.005.

131 DEMIR, Özlem Tuğfe; BJÖRNSON, Emil; SANGUINETTI, Luca. Foundations of User-Centric Cell-Free Massive MIMO. **Foundations and Trends in Signal Processing**, Now Publishers, v. 14, n. 3–4, p. 162–472, 2021. ISSN 1932-8346. DOI: 10.1561/2000000109.

132 VU, Tung T. *et al.* Cell-Free Massive MIMO for Wireless Federated Learning. **IEEE Transactions on Wireless Communications**, v. 19, n. 10, p. 6377–6392, 2020. DOI: 10.1109/TWC.2020.3002988.

133 VU, Tung T. *et al.* Joint Resource Allocation to Minimize Execution Time of Federated Learning in Cell-Free Massive MIMO. **IEEE Internet of Things Journal**, v. 9, n. 21, p. 21736–21750, 2022. DOI: 10.1109/JIOT.2022.3183295.

134 SIFAOU, Houssem; LI, Geoffrey Ye. Over-the-Air Federated Learning Over Scalable Cell-Free Massive MIMO. **IEEE Transactions on Wireless Communications**, v. 23, n. 5, p. 4214–4227, 2024. DOI: 10.1109/TWC.2023.3315962.

135 NUNES, Alan L. *et al.* Optimizing Computational Costs of Spark for SARS-CoV-2 Sequences Comparisons on a Commercial Cloud. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 35, n. 18, e7678, 2023. DOI: 10.1002/cpe.7678.

136 NUNES, Alan L. *et al.* Two-Step Estimation Strategy for Predicting Petroleum Reservoir Simulation Jobs Runtime on an HPC Cluster. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 37, n. 4–5, e70026, 2025. DOI: 10.1002/cpe.70026.

137 CAMPBELL, R. *et al.* MapReduce na AWS: Uma Análise de Custos Computacionais Utilizando os Serviços FaaS e IaaS. *In:* ANAIS do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD). Florianópolis/SC, Brazil: Sociedade Brasileira de Computação, 2022. p. 145–156. DOI: 10.5753/wscad.2022.226308.

138 NUNES, Alan L.; DRUMMOND, Lúcia M. A.; BOERES, Cristina. Reduzindo Custos de Implantação e Execução de Clusters Spark em Nuvens Públicas. *In:* ANAIS da VIII Escola Regional de Alto Desempenho do Rio de Janeiro (ERAD-RJ). Rio de Janeiro/RJ, Brazil: Sociedade Brasileira de Computação, 2023. p. 6–10. DOI: 10.5753/eradrj.2023.231713.

139 LOPES, C. *et al.* Flower-PROV: Captura Distribuída de Dados de Proveniência em Experimentos de Aprendizado Federado. *In:* ANAIS Estendidos do XXXVIII Simpósio Brasileiro de Banco de Dados (SBBD). Belo Horizonte/MG, Brazil: Sociedade Brasileira de Computação, 2023. p. 90–95. DOI: 10.5753/sbbd_estendido.2023.233337.

140 NUNES, Alan L. *et al.* Prediction of Reservoir Simulation Jobs Times Using a Real-World SLURM Log. *In:* ANAIS do XXIV Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD). Porto Alegre/RS, Brazil: Sociedade Brasileira de Computação, 2023. p. 49–60. DOI: 10.5753/wscad.2023.235649.

141 LOPES, Camila R. *et al.* Provenance-Based Dynamic Fine-Tuning of Cross-Silo Federated Learning. *In:* HIGH Performance Computing: 10th Latin American Conference (CARLA), Revised Selected Papers. Cartagena, Colombia: Springer, 2024. v. 1887. (Communications in Computer and Information Science). p. 113–127. DOI: 10.1007/978-3-031-52186-7_8.

142 NUNES, Alan L. *et al.* A Framework for Executing Long Simulation Jobs Cheaply in the Cloud. *In:* 2024 IEEE International Conference on Cloud Engineering (IC2E). Paphos, Cyprus: IEEE, 2024. p. 233–244. DOI: 10.1109/IC2E61754.2024.00033.

143 LIMA, M. de *et al.* Modelos de Predição do Tempo de Jobs Aplicados a um Ambiente de Produção de Alto Desempenho. *In:* ANAIS do XXV Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD). São Carlos/SP, Brazil: Sociedade Brasileira de Computação, 2024. p. 25–36. DOI: 10.5753/sscad.2024.244537.

144 NUNES, Alan L. *et al.* Explorando Modelos Preditivos de Tempo de Execução de Simulações de Reservatório em Clusters HPC. *In:* ESCOLA Regional de Alto Desempenho da Região Sudeste (ERAD-SE). Niterói/RJ, Brazil: Sociedade Brasileira de Computação, 2025. p. 6–10. DOI: 10.5753/eradse.2025.16931.

145 PORTELLA, Felipe A. *et al.* A Dataset of Petroleum Reservoir Simulations in a High-Performance Computing Cluster. Submitted to Data in Brief, under review., 2026.

146 NUNES, Alan L. *et al.* Wise-Burst: A Decision Support Tool for Cloud Bursting. Submitted to IEEE Transactions on Cloud Computing, under review., 2026.

147 NUNES, Alan L. *et al.* A Reliability-Aware Client Selection Framework for Federated Learning on Heterogeneous Resources under Dynamic Availability. Submitted to the 32nd IEEE International Conference on Parallel and Distributed

Systems (ICPADS), under review. Preprint available at hal.science/hal-05663449., 2026.

148 NUNES, Alan L. *et al.* **Optimal Client Selection Algorithms for Federated Learning**. Atlanta, GA, United States: IEEE Press, 2024. Research poster presented at SC24: The International Conference for High Performance Computing, Networking, Storage and Analysis. Available from: https://sc24.supercomputing.org/proceedings/poster/poster_pages/post232.html.

149 NUNES, Alan L. *et al.* **Optimal Client Selection Algorithms for Federated Learning**. Talence, France: HAL Open Science, 2025. Poster presented at Journée de l'EDMI 2025 – Journée annuelle de l'école doctorale de mathématiques et d'informatique. Available from: https://hal.science/hal-05632142.

150 NUNES, Alan L. **Optimal Time and Energy-Aware Client Selection Algorithms for Federated Learning on Heterogeneous Resources**. Nantes, France: IETR/LS2N/Nantes Université, 2024. Abstract presented at Compas'2024: Conférence d'Informatique en Parallélisme, Architecture et Système. Available from: https://crp.info.ucl.ac.be/compas2024/doc/compas2024-paper19.pdf?cap=hcav19SXnCNnzJKjGngoqQfoxfeshC.

151 NUNES, Alan L.; LIMA PILLA, Laércio; DRUMMOND, Lucia. **Scheduling Algorithms for the Optimization of Distributed Machine Learning Models on Heterogeneous Resources**. Talence, France: HAL Open Science, 2026. Extended abstract prepared for Journée de l'EDMI 2026 – Journée annuelle de l'école doctorale de mathématiques et d'informatique. Available from: https://hal.science/hal-05632171.

152 COVER, Thomas M.; THOMAS, Joy A. **Elements of Information Theory**. 2. ed. Hoboken, NJ, United States: Wiley-Interscience, 2006. ISBN 978-0-471-24195-9. DOI: 10.1002/047174882X.

153 ROSS, Sheldon M. **Introduction to Probability Models**. 11. ed. Amsterdam, Netherlands: Academic Press, 2014. ISBN 978-0-12-407948-9.

APPENDIX A — Scientific Contributions

Throughout the PhD project, I was involved in a range of research, collaboration, and dissemination activities, including journal articles (135, 136, 29), conference papers (137, 138, 139, 140, 141, 142, 143, 28, 144), submitted manuscripts (145, 146, 147), posters (148, 149), and extended abstracts (150, 151). These outputs, developed between 2022 and 2026, cover topics such as federated learning, client selection, scheduling, provenance, execution-time prediction, and cloud cost optimization. Although not all of these collaborations are directly central to this thesis, they expanded my research experience across distributed systems, high-performance computing, and applied machine learning.

Among these contributions, the works most closely aligned with the thesis subject address client selection, scheduling, and resource-aware optimization in federated learning. The core contribution is the SBAC-PAD 2024 paper (28), which introduces the MEC and ECMTC algorithms for jointly optimizing training time and energy consumption on heterogeneous resources. MEC prioritizes the minimization of the training round duration, whereas ECMTC prioritizes energy consumption under a time constraint. This line of work was also disseminated through the SC'24 poster (148), the Compas'2024 abstract (150), and the Journée de l'EDMI 2025 poster (149).

MetaCS-FL (29) extends this research direction by generalizing client selection through a metaheuristic-based framework for multi-objective optimization. While MEC and ECMTC provide exact solutions for specific time–energy objective orders, MetaCS-FL targets more flexible FL scenarios in which optimization priorities may vary according to the application. The extended abstract prepared for Journée de l'EDMI 2026 (151) is related to this manuscript, summarizing the same research direction from the perspective of scheduling algorithms for distributed machine learning models on heterogeneous resources. The extension of MetaCS-FL submitted to ICPADS 2026 (147) introduces reliability-aware client selection under dynamic availability of clients, considering late-joining and intermittent-availability scenarios. It adjusts client workloads according to availability and completion histories, reducing the impact of unreliable devices while preserving their participation.

APPENDIX B — Reproducibility and Implementation Details

This appendix complements the experimental evaluation in Chapter 6 by detailing the implementation choices and reproducibility information underlying the experiments. It is organized as follows. Section B.1 describes the adaptations made to the Flower framework. Section B.2 details the host profiling procedure, emulated devices, and FLOP-based scaling model used to approximate heterogeneous client behavior. Sections B.3 through B.6 then present the network simulation, client performance scoring, client availability simulation, and client cost estimation procedures. Finally, Sections B.7 and B.8 describe the dataset preparation procedure, model architectures, and training hyperparameters used.

B.1 Flower Framework Adaptations

The experimental implementation is built on top of *Flower* (53), a federated learning framework that provides a convenient abstraction for implementing clients, server-side strategies, and the communication between the server and clients. Flower was particularly useful in this work because it already implements the lower-level communication layer through gRPC channels, allowing the implementation to focus on the client selection logic, profiling procedure, cost estimation, and metric collection.

Nevertheless, the default Flower execution model was not sufficient to support the full behavior required by *MetaCS-FL*'s system model (Sections 5.1.1 and 5.1.2). In particular, the standard Flower server loop is primarily configured around a fixed number of FL rounds. In contrast, the experiments in this work require more flexible stopping criteria, such as stopping when a target testing accuracy is reached or when a maximum number of rounds is completed (Section 6.1.5). For this reason, the server-side strategy was extended with an application-level stopping mechanism that evaluates the configured criteria during Flower's FL loop and terminates the FL execution once the stopping criteria are satisfied.

A second limitation concerns client profiling. The proposed framework requires performance information before client selection can be performed reliably (Section 5.1.2). Therefore, the implementation distinguishes two profiling cases. At the beginning of the execution, when no client profile is available, the server performs a cold-start profiling stage. This stage is blocking: the FL loop waits until the initially available clients have completed profiling, ensuring that the first client selection step has access to the required performance estimates. Later, if new clients join the system during the FL loop, they are profiled asynchronously. This non-blocking profiling mechanism allows late-joining clients to be characterized without interrupting the ongoing FL execution. Once their profiles become available, they can be considered in subsequent client selection decisions.

A third required adaptation concerns client identity. Flower internally manages connected clients through client proxy objects. These proxy identifiers are useful for communication, but they are not sufficient for this work because the client selection algorithms must select concrete logical clients, such as `client_0`, `client_1`, and so forth. Without explicitly associating each client proxy with the corresponding logical client identifier, the server cannot guarantee that the exact clients selected by the optimization algorithm are the ones instructed to train or test in the next round. To address this, the modified server (see `flwr/server/server.py`) queries each client for its logical identifier using `get_properties` and propagates this identifier through the fit and evaluation results. This makes it possible to map Flower client proxies to the logical client identifiers used by the selection algorithms.

In addition, the gRPC server was adjusted to handle the scale and duration of the experiments (see `flwr/server/superlink/fleet/grpc_bidi/grpc_server.py`). Specifically, the maximum number of concurrent workers was increased from 1,000 to 10,000, and the keepalive time was extended from 210,000 ms to 7,200,000 ms. These changes reduce the risk of interrupting long-running client connections during large experiments.

These adaptations preserve the main Flower execution model while extending it with the mechanisms required by this work: flexible FL stopping criteria, blocking profiling for initially available clients, non-blocking profiling for late-joining clients, explicit mapping between logical client identifiers and Flower client proxies, and communication settings suitable for large-scale experiments.

B.2 Device Profiling and Performance Scaling

B.2.1 Host profiling procedure

Before each client is launched, the framework performs a controlled profiling procedure on the host machine where the client process is executed. This procedure is implemented by the `HostProfiler` component and characterizes the execution environment available to the client before the FL workload starts.

For each client, the executor creates a `HostProfiler` instance using the current execution output directory and invokes `profile_host_n_times`. The number of repetitions is controlled by the configuration parameter `num_host_profiles`. Each repetition collects CPU information, sustained compute throughput, memory bandwidth, storage throughput, and power-related measurements. The collected profiles are averaged recursively across numeric fields, rounded to two decimal places, saved as a JSON file, and returned to the client launcher as part of the client-specific configuration.

The CPU profile includes the processor model, number of sockets, number of cores per socket, number of logical threads, and maximum CPU frequency. These values are obtained using `lscpu` and `psutil`. The compute profile estimates sustained floating-point throughput by multiplying two dense 2000×2000 matrices using NumPy:

$$\text{GFLOP/s} = \frac{2 \cdot n^3}{t \cdot 10^9} \tag{B.1}$$

where $n = 2000$ and t is the measured matrix multiplication time in seconds. The number of OpenBLAS threads is set to the number of physical CPU cores.

The memory profile estimates memory bandwidth using the `stress-ng` stream benchmark with four workers and a 10-second timeout. If `stress-ng` is unavailable, a fallback value of 5 GB/s is used. The storage profile estimates sequential write throughput using `fio`, writing a 512 MB temporary file with 1 MB blocks for five seconds. If `fio` is unavailable, a fallback value of 100 MB/s is used.

The power profile is obtained using `pyRAPL` when available. In this case, package-level energy is recorded over a three-second interval. If `pyRAPL` is unavailable or fails, the profiler falls back to an approximate CPU-utilization-based estimate.

After profiling, the averaged host measurements are written to the `host_profile` directory within the execution output folder. Each profile is saved as a JSON file named according to the pattern `{hostname}_client_{client_id}.json`. As a result, the frame-

work keeps a separate profiling record for each client, including cases where multiple client processes run on the same physical node.

Host profiling is distinct from device emulation. Host profiling characterizes the physical machine executing the client process, whereas device emulation defines the target edge-device profile used to approximate heterogeneous client behavior.

B.2.2 Emulated device specifications

To emulate heterogeneous edge environments, each FL client is assigned a 4-core target device profile before execution. The evaluated profiles include Raspberry Pi, NVIDIA Jetson, Intel Atom, Qualcomm Snapdragon, and Google Coral Edge TPU devices.

Each device profile specifies CPU, memory, power, and battery-related parameters. The CPU parameters include the number of cores, CPU frequency, peak CPU throughput, and effective floating-point operations per CPU cycle per core (FPOCC) for training and inference. The memory parameters include memory size and memory bandwidth. The power parameters include the mean power consumption during data reception, computation, transmission, and idle periods. The battery parameters include capacity, voltage, type, and maximum stored energy. Table 39 summarizes the main CPU, memory, and power specifications used by the emulated devices.

Table 39: Summary of emulated device specifications.

Device	CPU				Memory		Power		
	Cores	Freq. (GHz)	Peak (GFLOP/s)	Train FPOCC	Size (GB)	BW (GB/s)	P_{rx} (W)	P_{comp} (W)	P_{tx} (W)
Google Coral Edge TPU Cortex-A53	4	1.20	19.20	1.6–3.0	4	12.8	4.5	6.0	5.0
Intel Atom x5-Z8350	4	1.44	23.04	1.4–2.6	4	12.8	4.5	6.0	5.0
Intel Atom x7-E3950	4	2.00	32.00	1.6–2.8	8	29.8	7.0	10.0	8.0
NVIDIA Jetson Nano	4	1.43	22.88	1.6–2.8	4	25.6	5.0	10.0	7.0
NVIDIA Jetson TX1	4	1.90	60.80	3.2–6.0	4	25.6	7.0	10.0	8.0
NVIDIA Jetson TX2	4	2.00	128.00	6.4–12.8	8	29.8	8.0	15.0	10.0
Qualcomm Snapdragon 410E	4	1.20	19.20	1.4–2.4	2	4.2	2.8	4.0	3.2
Qualcomm Snapdragon 820E	4	2.20	70.40	3.2–6.0	4	29.8	5.5	7.0	6.0
Raspberry Pi 3	4	1.20	9.60	0.6–1.2	1	7.2	2.5	3.5	2.8
Raspberry Pi 4	4	1.50	24.00	1.6–2.8	4	25.6	4.8	6.4	5.2

The peak CPU throughput of a device is computed as

$$\text{Peak GFLOP/s} = \frac{N_{\text{cores}} \cdot f_{\text{CPU}} \cdot L_{\text{SIMD}} \cdot U_{\text{FMA}} \cdot I_{\text{FMA}}}{10^9} \tag{B.2}$$

where N_{cores} is the number of CPU cores, f_{CPU} is the CPU frequency in hertz, L_{SIMD} is the number of SIMD lanes, U_{FMA} is the number of FMA units, and I_{FMA} is the number of

FMA instructions issued per cycle. The division by 10^9 converts the resulting value from FLOP/s to GFLOP/s.

The maximum stored battery energy is computed as

$$E_{\max} = V \cdot \frac{C_{\mathrm{mAh}}}{1000} \cdot 3600 \tag{B.3}$$

where $E_{\max}$ is the maximum stored battery energy in joules, V is the battery voltage in volts, and C_{mAh} is the battery capacity in milliampere-hours. The factor 1000 converts milliampere-hours to ampere-hours, and the factor 3600 converts watt-hours to joules.

In Table 39, *Cores* denotes the number of emulated CPU cores, *Freq.* denotes CPU frequency in GHz, *Peak* denotes peak CPU throughput in GFLOP/s, and *Train FPOCC* denotes the training-phase FPOCC range sampled by the client. Under memory, *Size* denotes memory capacity in GB and *BW* denotes memory bandwidth in GB/s. Under power, P_{rx}, P_{comp}, and P_{tx} denote the mean power consumption during data reception, computation, and data transmission, respectively.

For 50-client systems, the client-resource mapping contains client identifiers from 0 to 49. For 100-client systems, it contains identifiers from 0 to 99. The mapping is generated before execution and remains fixed throughout the run. It is stored in `clients_resources.csv`, which records the assigned device, network profile, geographic location, initial battery energy, failure probability, and device score for each client.

B.2.3 FLOP-based host-to-device scaling model

The client-side implementation estimates the computation time that a training or testing workload would require on an assigned emulated device using a FLOP-based host-to-device scaling model. This model is applied when clients execute profiling or regular FL workloads. The resulting emulated-device computation time and FLOP-related metrics are stored in the client metrics and later reused by the server during cost-matrix generation.

During training, the client launches the local training task in a separate process and records the actual host computation time. If Linux `perf` is available, hardware performance counters are collected while the training process is running. After training completes, the client reloads the local model, builds a concrete TensorFlow training function using a dummy batch with the same input shape and batch size as the real workload, and applies TensorFlow's graph profiler to estimate the floating-point operations required by

one batch. The profiled training function includes the forward pass, loss computation, gradient computation, and optimizer update.

During testing, the same procedure is applied to the local evaluation task. The client launches testing in a separate process, optionally collects `perf` counters, records the actual host computation time, builds a concrete inference function using a dummy batch, and profiles the corresponding forward pass.

Let i denote a client, r denote the FL communication round, $p \in \{\text{train}, \text{test}\}$ denote the FL phase, and m denote the model being trained or evaluated. The TensorFlow profiler returns the number of floating-point operations for one profiled batch. The implementation then derives FPO_m^p by normalizing this batch-level count by the batch size, so that FPO_m^p represents a per-sample estimate. From this profiled batch cost, the client computes the number of floating-point operations per sample and the total number of floating-point operations required by the local workload:

$$\text{TFPO}_{i,r}^p = \text{FPO}_m^p \cdot b_i^p \cdot \left\lceil \frac{d_i^p}{b_i^p} \right\rceil \cdot e_i^p \tag{B.4}$$

where $\text{TFPO}_{i,r}^p$ is the total number of floating-point operations performed by client i in round r and phase p, FPO_m^p is the number of floating-point operations per sample for model m in phase p, b_i^p is the batch size used by client i, d_i^p is the number of local samples processed by client i, and e_i^p is the number of local epochs. The term $b_i^p \cdot \lceil d_i^p / b_i^p \rceil$ corresponds to the effective number of samples processed after accounting for the last possibly incomplete batch. For testing, $e_i^p = 1$ because evaluation performs a single pass over the assigned test samples.

The client stores the quantities required by the scaling model in `comp_metrics.csv`. For training, these include `fpo_train_m`, `total_float_ops`, `bs_train_i`, `ds_train_i`, `e_i`, `nc_cpu_i`, `fq_cpu_i`, `fpocc_m_i_train`, `bw_mem_i`, and `mt_m_train`. Testing stores the analogous testing fields, including `fpo_test_m`, `bs_test_i`, `ds_test_i`, `fpocc_m_i_test`, and `mt_m_test`. In these field names, `bs` denotes batch size, `ds` denotes dataset size, `nc_cpu` denotes the number of emulated CPU cores, `fq_cpu` denotes emulated CPU frequency, `fpocc` denotes floating-point operations per CPU cycle per core, `bw_mem` denotes memory bandwidth, and `mt_m` denotes memory traffic per floating-point operation.

The FLOP-based scaling approach uses the measured host computation time to estimate how efficiently the workload executed on the host. Let $T_{i,r,\text{host}}^p$ denote the actual host computation time measured by client i in round r and phase p. The achieved host

throughput is computed as

$$G^{p}_{i,r,\text{host}} = \frac{\text{TFPO}^{p}_{i,r}}{T^{p}_{i,r,\text{host}} \cdot 10^{9}} \tag{B.5}$$

where $G^{p}_{i,r,\text{host}}$ is the achieved host throughput in GFLOP/s. The factor 10^9 converts FLOP/s to GFLOP/s.

The achieved host throughput is then compared with the sustained host throughput measured during host profiling:

$$\eta^{p}_{i,r,\text{host}} = \frac{G^{p}_{i,r,\text{host}}}{G^{\text{sustained}}_{\text{host}}} \tag{B.6}$$

Here, $\eta^{p}_{i,r,\text{host}}$ is the host execution efficiency of client i in round r and phase p, and $G^{\text{sustained}}_{\text{host}}$ is the sustained host throughput measured during host profiling, also expressed in GFLOP/s.

To account for the typically lower practical efficiency of constrained edge devices, the device efficiency is approximated as

$$\eta^{p}_{i,r,\text{dev}} = 0.8 \cdot \eta^{p}_{i,r,\text{host}} \tag{B.7}$$

where $\eta^{p}_{i,r,\text{dev}}$ is the estimated execution efficiency of the emulated device assigned to client i. The constant 0.8 is a correction factor used to reduce the host efficiency when transferring it to the emulated edge device.

Finally, the emulated computation time is estimated by dividing the total number of floating-point operations by the effective throughput of the assigned device:

$$\widehat{T}^{p}_{i,r,\text{dev}} = \frac{\text{TFPO}^{p}_{i,r}}{G^{\text{peak}}_{i} \cdot 10^{9} \cdot \eta^{p}_{i,r,\text{dev}}} \tag{B.8}$$

where $\widehat{T}^{p}_{i,r,\text{dev}}$ is the estimated computation time on the emulated device, G^{peak}_{i} is the peak CPU throughput of the emulated device assigned to client i in GFLOP/s, and $\eta^{p}_{i,r,\text{dev}}$ is the estimated device execution efficiency. The term $G^{\text{peak}}_{i} \cdot 10^{9} \cdot \eta^{p}_{i,r,\text{dev}}$ therefore represents the effective device throughput in FLOP/s.

B.3 Network Simulation

The network simulation component models heterogeneous and time-varying communication conditions between the FL server and clients. Before execution, each client is assigned a network profile specifying bandwidth statistics, latency, jitter, packet loss, maximum

segment size, and geographic locations.

In the evaluated experiments, the server is placed in Paris, while clients are sampled from 12 possible locations: ten in France (Bordeaux, Grenoble, Lille, Lyon, Nancy, Nantes, Rennes, Sophia Antipolis, Strasbourg, and Toulouse) and two in neighboring countries (Louvain in Belgium and Luxembourg City in Luxembourg). The experiments use 4G LTE network profiles with three quality scenarios: excellent, moderate, and poor.

Figure 32 illustrates the geographical distribution of the server and the possible client locations considered in the experiments.

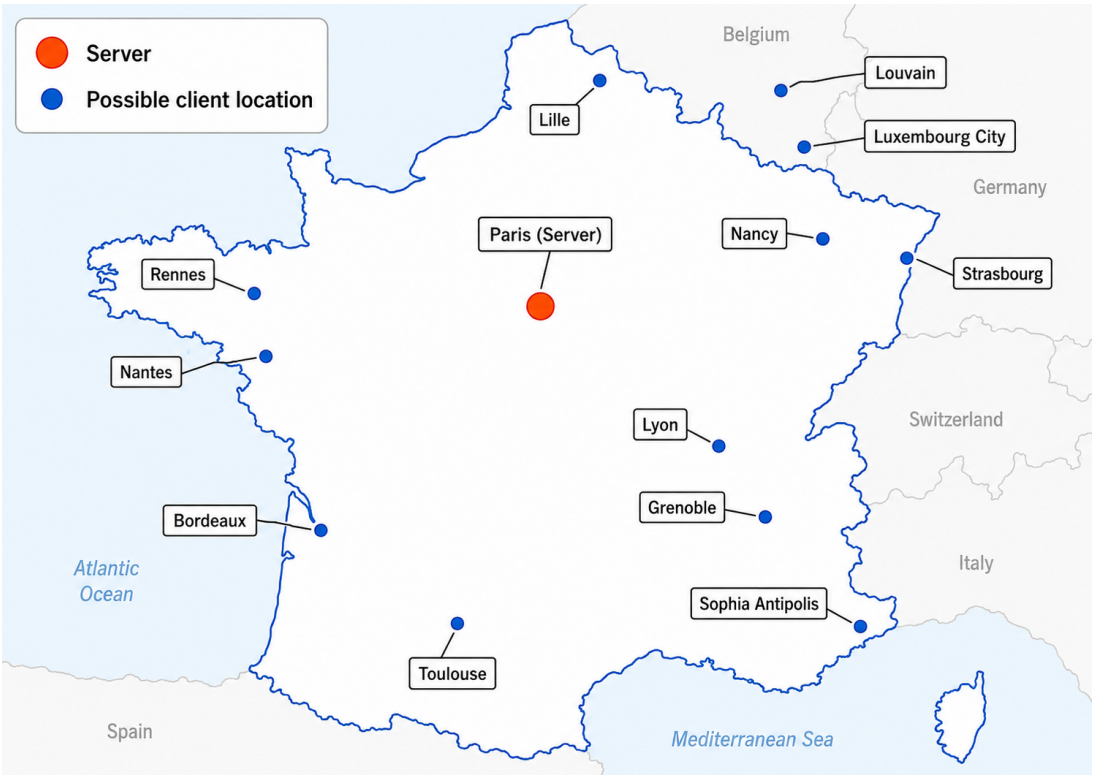

Figure 32: Geographical distribution of the FL server and possible client locations used in the network simulation. The server is located in Paris, while client locations are sampled from ten cities in France and two cities in neighboring countries, namely Louvain (Belgium) and Luxembourg City (Luxembourg).

B.3.1 Autoregressive network model

Upload and download bandwidths are modeled using an autoregressive process. Each client maintains independent upload and download histories, initialized from normal distributions centered at the corresponding mean bandwidths.

Let i denote a client, t denote a communication event, and $q \in \{u,d\}$ denote the communication direction, where u represents upload and d represents download. Let $B^u_{i,t}$

and $B^d_{i,t}$ denote the upload and download bandwidths of client i at event t, respectively. For each direction q, the next bandwidth value is computed as

$$B^q_{i,t} = \mu^q_i + \sum_{k=1}^{P_{\mathrm{AR}}} \phi_k \cdot \left(B^q_{i,t-k} - \mu^q_i\right) + \varepsilon^q_{i,t} \tag{B.9}$$

where μ^q_i is the mean bandwidth assigned to client i in direction q, P_{AR} is the autoregressive order, ϕ_k is the k-th autoregressive coefficient, and $\varepsilon^q_{i,t}$ is a Gaussian noise term sampled from $\mathcal{N}(0,\sigma^q_i)$. The parameter σ^q_i is the bandwidth standard deviation assigned to client i in direction q. The implementation uses an AR(2) process, *i.e.*, $P_{\mathrm{AR}} = 2$, with coefficients $\phi = [0.5,0.3]$.

A lower bound is applied to prevent unrealistically small bandwidth values:

$$\widetilde{B}^q_{i,t} = \max\left(B^q_{i,t}, B^q_{i,\,\min}\right) \tag{B.10}$$

where $\widetilde{B}^q_{i,t}$ is the bounded bandwidth value used by the simulator and $B^q_{i,\,\min}$ is the minimum allowed bandwidth for client i in direction q.

Latency is modeled by adding uniformly sampled jitter to the base latency:

$$L_{i,t} = \max\left(0, L^{\mathrm{base}}_i + J_{i,t}\right), \qquad J_{i,t} \sim \mathcal{U}(-j_i,j_i) \tag{B.11}$$

Here, $L_{i,t}$ is the sampled latency of client i at communication event t, L^{base}_i is the base latency assigned by the client's network profile, $J_{i,t}$ is the sampled jitter term, and j_i is the maximum jitter magnitude.

Packet loss is modeled as a Bernoulli event:

$$X_{i,t} \sim \mathrm{Bernoulli}(\rho_i) \tag{B.12}$$

where $X_{i,t}$ and ρ_i are the packet-loss indicator at event t and packet-loss rate for client i, respectively.

B.3.2 Bandwidth, latency, packet loss, and jitter parameters

Although the framework supports 3G, 4G LTE, 5G, Wi-Fi, and fixed broadband profiles, the experiments reported in this work use 4G LTE. The base 4G LTE profile assumes 10 Mbps upload bandwidth, 50 Mbps download bandwidth, and 50 ms base latency. Table 40 summarizes the parameters used for the excellent, moderate, and poor network scenarios.

The latency jitter values are 5 ms, 10 ms, and 15 ms for the excellent, moderate,

Table 40: 4G LTE network parameters used in the experiments.

Network Scenario	Upload Bandwidth			Download Bandwidth			Latency (ms)	Loss
	Mean (Mbps)	Std. (Mbps)	Min. (Mbps)	Mean (Mbps)	Std. (Mbps)	Min. (Mbps)		
Excellent	9.0	0.5	7.0	45.0	2.5	35.0	40.0	0.001
Moderate	6.0	1.0	4.0	30.0	5.0	20.0	50.0	0.010
Poor	3.0	1.5	1.0	15.0	7.5	5.0	75.0	0.050

and poor scenarios, respectively. All profiles assume an MTU of 1500 bytes, resulting in maximum segment sizes of 1460 bytes for IPv4 and 1440 bytes for IPv6.

B.3.3 Geographic client placement

To incorporate distance-based latency, each client is assigned a geographic location before execution. The server location is fixed to Paris, while each client location is sampled from the candidate city list.

Table 41 reports the candidate client locations and their distances to the server, computed using the Haversine formula. Let (φ_i,λ_i) denote the latitude and longitude of client i, and let (φ_s,λ_s) denote the latitude and longitude of the server. All angular coordinates are expressed in radians. The great-circle distance is computed as

$$d_i = 2 \cdot R \cdot \arctan 2\left(\sqrt{a_i}, \sqrt{1-a_i}\right) \tag{B.13}$$

where d_i is the distance between client i and the server, $R = 6371$ km is the Earth radius, and a_i is the Haversine intermediate term:

$$a_i = \sin^2\left(\frac{\varphi_i - \varphi_s}{2}\right) + \cos(\varphi_s) \cdot \cos(\varphi_i) \cdot \sin^2\left(\frac{\lambda_i - \lambda_s}{2}\right) \tag{B.14}$$

In the implementation, the network simulator first samples an instantaneous latency by adding jitter to the client's base latency. This instantaneous latency is then passed to the transfer-time model, which combines it with the distance-based propagation component derived from the client and server locations. The distance-adjusted round-trip latency is represented as

$$L_i^{\text{dist}} = L_i + \frac{2 \cdot d_i}{v} \tag{B.15}$$

where L_i^{dist} is the distance-adjusted latency, L_i is the instantaneous latency after applying jitter, d_i is the client–server distance, and $v = 200$ km/ms approximates the speed of light

in fiber optics. The factor of two accounts for the round-trip path between the client and the server.

Table 41: Candidate geographic locations used for client placement.

Server Location	Client Location	Distance to Server (km)
Paris	Bordeaux	499.3
	Grenoble	481.2
	Lille	203.5
	Louvain	280.9
	Luxembourg	286.9
	Lyon	391.5
	Nancy	281.4
	Nantes	342.7
	Rennes	308.1
	Sophia	686.4
	Strasbourg	397.3
	Toulouse	588.1

The sampled location of each client is written to `clients_resources.csv` together with the assigned network and device parameters.

B.4 Client Performance Scoring

The late-joining experiments (Section 6.3.1) account for differences in the system capabilities of clients that become available after training has started. After each client's emulated device, network profile, and geographic location are sampled, an overall performance score is assigned and used to define the *worst* or *best* late-joining clients, while the client-resource mapping remains fixed throughout the execution.

Let $\mathcal{C}$ denote the set of clients in an execution. For each client $i \in \mathcal{C}$, three partial scores are computed: computation, network, and energy. These scores are normalized using percentile-based baselines from the same execution: the 90th percentile for characteristics where higher values are better, such as CPU capability, memory size, and bandwidth, and the 10th percentile for characteristics where lower values are better, such as latency, jitter, packet loss, and power consumption.

Each client is scored along three dimensions: computation, network, and energy. The corresponding raw characteristics are normalized against percentile-based baselines computed from the clients in the same execution, producing partial scores in the range

[0,100]. To cap normalized values at one, we define the clipping function $c(x) = \min(x,1)$.

For resource characteristics where larger values are preferable, such as CPU capability, memory size, and bandwidth, the normalization uses the 90th percentile. For cost-like characteristics where smaller values are preferable, such as latency, jitter, packet loss, and power consumption, the normalization uses the 10th percentile.

The computation score captures the processing and memory capacity of client i. It combines the mean training FPOCC, the CPU capacity, and the memory size:

$$S_i^{\text{comp}} = 100 \cdot \left(0.45 \cdot c\left(\frac{\overline{\text{FPOCC}}_i}{Q_{90}(\overline{\text{FPOCC}})} \right) + 0.35 \cdot c\left(\frac{\text{CPUPerf}_i}{Q_{90}(\text{CPUPerf})} \right) + 0.20 \cdot c\left(\frac{M_i}{Q_{90}(M)} \right) \right) \tag{B.16}$$

where

$$\overline{\text{FPOCC}}_i = \frac{\text{FPOCC}_{i,\,\min}^{\text{train}} + \text{FPOCC}_{i,\,\max}^{\text{train}}}{2}, \qquad \text{CPUPerf}_i = N_i^{\text{cores}} \cdot \frac{f_i^{\text{CPU}}}{10^9} \tag{B.17}$$

Here, M_i is the memory capacity, N_i^{cores} is the number of emulated CPU cores, and f_i^{CPU} is the CPU frequency in hertz.

The network score captures both throughput and communication stability. Higher upload and download bandwidths increase the score, whereas lower latency, jitter, and packet loss are favored:

$$\begin{aligned} S_i^{\text{net}} = 100 \cdot \Bigg(& 0.30 \cdot c\left(\frac{B_i^u}{Q_{90}(B^u)} \right) + 0.25 \cdot c\left(\frac{B_i^d}{Q_{90}(B^d)} \right) + 0.25 \cdot c\left(\frac{Q_{10}(L)}{L_i} \right) \\ & + 0.10 \cdot c\left(\frac{Q_{10}(J)}{J_i} \right) + 0.10 \cdot c\left(\frac{Q_{10}(\rho)}{\rho_i} \right) \Bigg) \end{aligned} \tag{B.18}$$

where B_i^u and B_i^d are the mean upload and download bandwidths, L_i is the base latency, J_i is the latency jitter, and ρ_i is the packet-loss rate.

The energy score favors clients with lower power consumption during reception, computation, transmission, and idle periods:

$$S_i^{\text{energy}} = 25 \cdot \sum_{x \in \{\text{rx,comp,tx,idle}\}} c\left(\frac{Q_{10}(P^x)}{\max(P_i^x, 0.1)} \right) \tag{B.19}$$

where P_i^x denotes the mean power consumption of client i in state x. The lower bound of 0.1 W avoids divisions by very small values.

The overall client score is computed as:

$$S_i = 0.50 \cdot S_i^{\text{comp}} + 0.25 \cdot S_i^{\text{net}} + 0.25 \cdot S_i^{\text{energy}} \tag{B.20}$$

Computation is therefore weighted most heavily, while network and energy contribute equally. The clients are then sorted by increasing S_i, and the number of late-joining clients is determined from the configured percentage p_{late} as:

$$n_{\text{late}} = \min\left(\max\left(1, \lfloor |\mathcal{C}| \cdot p_{\text{late}} \rfloor\right), |\mathcal{C}|\right) \tag{B.21}$$

The *worst* profile selects the n_{late} lowest-scoring clients, whereas the *best* profile selects the n_{late} highest-scoring clients. The selected clients are then marked as late-joining and assigned the configured first appearance round. For reproducibility, the ordered score list is saved to `clients_scores.csv`, while the selected late-joining clients are saved to `late_join_clients_ids.csv` with their score, profile, and first appearance round. The full client-resource mapping is recorded in `clients_resources.csv`.

B.5 Client Availability Simulation

The intermittent-availability experiments (Section 6.3.2) use a client-side simulator to emulate temporary client departures, returns, and completion failures during FL execution. The simulator is used when the server queries client properties before selection, preserving the same execution behavior used in the other experiments: the server does not directly control client availability, but observes it through the client property-query mechanism.

Each client is assigned an independent two-state availability process. Let $A_{i,r} \in \{0,1\}$ denote the availability state of client i in communication round r, where $A_{i,r} = 1$ means that the client is available and $A_{i,r} = 0$ means that it is unavailable. The state evolves according to a two-state Markov process:

$$\Pr(A_{i,r} = 0 \mid A_{i,r-1} = 1) = p_i^{\text{off}} \tag{B.22}$$

$$\Pr(A_{i,r} = 1 \mid A_{i,r-1} = 0) = p_i^{\text{on}} \tag{B.23}$$

where p_i^{off} is the probability that an available client becomes unavailable, and p_i^{on} is the probability that an unavailable client becomes available again. Therefore, if a client is available in round $r-1$, it remains available in round r with probability $1-p_i^{\text{off}}$. Similarly, if it is unavailable in round $r-1$, it remains unavailable in round r with probability $1-p_i^{\text{on}}$.

This model introduces temporal persistence in client availability. Unlike a memoryless Bernoulli model, where each round is sampled independently, the Markov process makes the current availability state depend on the previous one. Consequently, clients can experience bursts of availability or unavailability, which better represent intermittent behavior in cross-device FL systems. The processes are independent across clients, so each client evolves according to its own transition probabilities and random seed.

At the beginning of the execution, all clients are configured as initially available in the intermittent-availability experiments. During profiling, availability simulation can be temporarily bypassed to ensure that the initial profiles are collected. After profiling, the simulator updates the availability state whenever the server queries the client's properties for a communication round. To avoid advancing the Markov process multiple times in the same round, the simulator caches the sampled state for each round and returns the same value if the client is queried again.

The implementation supports heterogeneous availability behavior by assigning each client to an availability profile. Each profile defines a range of values for p_i^{off} and p_i^{on}. For example, stable clients have low p_i^{off} and high p_i^{on}, whereas clients with bursty unavailability have lower p_i^{on}, making them more likely to remain unavailable for multiple consecutive rounds. The experiments consider two intermittent-availability modes, denoted as *moderate* and *severe*. Both modes use the same four availability profiles, namely *stable*, *intermittent*, *unstable*, and *bursty-off*, but differ in the share of clients assigned to each profile and in the completion-failure parameters. The moderate mode represents a scenario in which most clients are stable or moderately intermittent, whereas the severe mode shifts the population toward less reliable profiles and increases the probability of post-selection completion failures. Table 42 summarizes the share of clients assigned to each profile and the corresponding transition-probability ranges for both modes.

Table 42: Availability profiles used in the intermittent-availability experiments.

Profile	**Moderate Share**	**Severe Share**	$\boldsymbol{p_i^{\text{off}}}$ **Range**	$\boldsymbol{p_i^{\text{on}}}$ **Range**
Stable	0.25	0.10	$[0.01, 0.03]$	$[0.70, 0.95]$
Intermittent	0.50	0.40	$[0.05, 0.12]$	$[0.30, 0.60]$
Unstable	0.20	0.35	$[0.15, 0.30]$	$[0.20, 0.45]$
Bursty-off	0.05	0.15	$[0.08, 0.18]$	$[0.05, 0.20]$

When the server requests client properties, the client updates or retrieves its availability state for the current round and reports this value through the `client_available` property. The server then builds the candidate set using only clients that are currently

available and have valid profiles. Therefore, the availability simulator affects client selection through the observed availability status while preserving the same server-side selection interface used in the other experiments.

In addition to pre-selection availability, the intermittent-availability experiments include post-selection completion failures. Under this mechanism, a client available during the property-query step and selected for training or testing may still fail to complete its assigned processing. This mechanism is applied after client selection, whereas the Markov availability process is used before selection to construct the candidate set. Thus, the simulator distinguishes between two different events: a client that is unavailable before selection and therefore cannot be selected, and a selected client that was initially reachable but fails to complete the assigned workload.

The completion-failure probability is profile- and workload-dependent. Each availability profile defines a base failure probability, reflecting that clients with less stable availability patterns are also more likely to fail after selection. The probability also increases with the assigned workload, since larger assignments require the client to remain available and active for longer. For a selected client i in round r, the completion-failure probability is defined as:

$$p_{i,r}^{\text{fail}} = \min\left(p_{\max}^{\text{fail}}, p_i^{\text{base}} + \theta_i \cdot \frac{x_{i,r}^{\text{samples}}}{\max(\texttt{AC}_i) \cdot s_{\text{task}}}\right) \tag{B.24}$$

where p_i^{base} is the base failure probability associated with the client's availability profile, θ_i controls the sensitivity to the assigned workload, $x_{i,r}^{\text{samples}}$ is the number of samples assigned to client i in round r, s_{task} is the number of samples per task, and $p_{\max}^{\text{fail}}$ caps the maximum failure probability. Since $\texttt{AC}_i$ is expressed in tasks, the denominator $\max(\texttt{AC}_i)\cdot s_{\text{task}}$ converts the maximum nominal task capacity into samples. This model captures situations in which a client is initially reachable but later becomes unable to finish its workload due to disconnection, interruption, battery depletion, or another local condition.

In the implementation, the workload-dependent term is computed from the number of examples assigned to the selected client and the maximum nominal capacity of that client. Completion failures are not sampled during profiling rounds, ensuring that profiling can be completed even when availability and completion-failure simulation are enabled. For regular training and testing rounds, the failure event is sampled after client selection using a deterministic random seed derived from the client identifier, the communication round, and the execution phase. As a result, the simulation is reproducible, while training and testing failures may still be sampled independently because the phase is part of the seed.

Table 43 summarizes the completion-failure parameters used for each availability profile. The moderate mode caps the maximum completion-failure probability at $p_{\max}^{\text{fail}} = 0.30$, whereas the severe mode uses $p_{\max}^{\text{fail}} = 0.35$. This prevents the simulator from making any selected client unrealistically unlikely to complete its processing, while still allowing unstable and heavily loaded clients to have higher failure probabilities.

Table 43: Completion-failure parameters used in the intermittent-availability experiments.

Profile	**Moderate** p_i^{base}	**Severe** p_i^{base}	θ_i	$p_{\max}^{\text{fail}}$
Stable	0.005	0.010	0.010	0.30 / 0.35
Intermittent	0.020	0.035	0.030	0.30 / 0.35
Unstable	0.060	0.090	0.060	0.30 / 0.35
Bursty-off	0.100	0.140	0.080	0.30 / 0.35

The distinction between pre-selection unavailability and post-selection completion failure is relevant because the reliability mechanism of MetaCS-FL assigns different penalties to these events. Availability failures affect candidate-set construction: a joined client unavailable before scheduling receives the availability penalty λ_{avail}. In contrast, completion failures affect the execution outcome after selection: a selected client that does not complete its assigned processing receives the completion penalty λ_{compl}, which is configured to be at least as large as λ_{avail} because such failures consume a selection slot and may directly affect the current round. This distinction allows MetaCS-FL to use the reliability score to reduce the admissible task-capacity slots of clients with poor availability or completion history, rather than permanently removing them from future selections.

B.6 Client Cost Estimation

The client selection algorithms require, for each available client i and each feasible task assignment $ac \in \mathtt{AC}_i$, an estimate of the time and energy that would be required to execute the corresponding FL phase. These estimates form the vectors $\mathtt{T}_i$ and $\mathtt{E}_i$ used by the optimization problem (Section 5.1.3). In the implementation, the cost-estimation procedure is performed in two stages. First, clients execute profiling or regular FL workloads and report execution metrics obtained from the emulated-device model. Second, the server uses these reported metrics to build the cost matrices used by the client selection algorithms.

The server does not directly benchmark the emulated devices. Instead, the client-side implementation first estimates the computation time that the workload would take on the assigned emulated device using the FLOP-based host-to-device scaling model described

in Section B.2.3. The resulting computation time, together with the number of floating-point operations, batch size, number of processed samples, number of epochs, device CPU parameters, memory bandwidth, and memory-traffic estimates, is stored as part of the client metrics. These metrics are then sent to the server and reused during client selection.

B.6.1 Computation cost estimation

For each client i, phase $p \in \{\text{train}, \text{test}\}$, and communication round r, let $T^p_{i,r,\text{ref}}$ denote the reference computation time reported by the client. This reference value already corresponds to the emulated-device computation time obtained after applying the host-to-device FLOP-based scaling model. Let $\text{TFPO}^p_{i,r,\text{ref}}$ be the total number of floating-point operations associated with the workload that produced this reference measurement. When the server evaluates a candidate assignment $ac \in \texttt{AC}_i$, it first updates the number of samples in the corresponding phase and recomputes the total number of floating-point operations:

$$\text{TFPO}^p_i(ac) = \text{FPO}^p_m \cdot b^p_i \cdot \left\lceil \frac{ac}{b^p_i} \right\rceil \cdot e^p_i \tag{B.25}$$

where FPO^p_m is the number of floating-point operations per sample for model m in phase p, b^p_i is the batch size, and e^p_i is the number of local epochs. For testing, $e^p_i = 1$.

The server then estimates the computation time for the candidate assignment by scaling the reference emulated-device computation time according to the ratio between the candidate FLOP count and the reference FLOP count:

$$\widehat{T}^{C,p}_i(ac) = T^p_{i,r,\text{ref}} \cdot \frac{\text{TFPO}^p_i(ac)}{\text{TFPO}^p_{i,r,\text{ref}}} \tag{B.26}$$

Thus, the server uses the same FLOP-based cost model as the clients, but it does so from the already scaled reference metrics reported by the clients. This avoids re-executing the workload for every feasible task assignment while still preserving the relationship between workload size and computation cost. If a client has participated in the current phase in previous rounds, the server uses the most recent phase metrics available for that client. Otherwise, it falls back to the closest profiling record with respect to the client's maximum feasible task capacity.

B.6.2 Communication cost estimation

The communication components are recomputed at selection time using the client's most recent network state. Before building the cost matrices, the server obtains the current download bandwidth, upload bandwidth, and latency reported by each client and overwrites the corresponding fields in the stored client metrics. The download and upload costs are then computed once per client and phase and reused for all candidate task capacities. This reflects the current implementation, in which the communication payloads are determined by the model parameters, phase instructions, and phase metrics, rather than by the candidate number of scheduled samples. In contrast, the computation component is recomputed for each $ac \in \texttt{AC}_i$ because the candidate assignment changes the number of local samples and, consequently, the total FLOP count.

Packet-loss events are simulated during client-side communication measurements and may add retransmission time to the measured transfer cost. During server-side cost-matrix generation, the current implementation refreshes bandwidth and latency but does not resimulate packet-loss events for every candidate assignment. Therefore, packet-loss effects are captured through the client-reported communication metrics, while bandwidth and latency are explicitly updated before the cost matrices are built.

For a candidate assignment $ac \in \texttt{AC}_i$, the estimated download time is

$$\widehat{T}_i^{D,p} = \frac{B_i^{D,p}}{\mathrm{BW}_i^D} + A_i^{D,p} \cdot \mathrm{RTT}_i^D \tag{B.27}$$

where $B_i^{D,p}$ is the number of bytes downloaded by client i in phase p, BW_i^D is the current download bandwidth, $A_i^{D,p}$ is the estimated number of acknowledgments, and RTT_i^D is the current round-trip latency used for the download interaction. Similarly, the estimated upload time is

$$\widehat{T}_i^{U,p} = \frac{B_i^{U,p}}{\mathrm{BW}_i^U} + A_i^{U,p} \cdot \mathrm{RTT}_i^U \tag{B.28}$$

where $B_i^{U,p}$ is the number of bytes uploaded by the client, BW_i^U is the current upload bandwidth, $A_i^{U,p}$ is the estimated number of acknowledgments, and RTT_i^U is the current round-trip latency used for the upload interaction.

B.6.3 Time estimation

The total estimated time for client i and assignment ac is then

$$\widehat{T}_i^p(ac) = \widehat{T}_i^{D,p} + \widehat{T}_i^{C,p}(ac) + \widehat{T}_i^{U,p} \tag{B.29}$$

For a fixed client and phase, the download and upload times are computed once using the current network state and reused for all $ac \in \mathtt{AC}_i$. The computation time is recomputed for each candidate assignment since it depends directly on the number of scheduled tasks.

B.6.4 Energy consumption estimation

Energy consumption is estimated using the same linear time–power model used in the system model. The download, computation, and upload energy components are

$$\widehat{E}_i^{D,p} = \widehat{T}_i^{D,p} \cdot P_i^{\mathrm{rx}}, \tag{B.30}$$

$$\widehat{E}_i^{C,p}(ac) = \widehat{T}_i^{C,p}(ac) \cdot P_i^{\mathrm{comp}}, \tag{B.31}$$

$$\widehat{E}_i^{U,p} = \widehat{T}_i^{U,p} \cdot P_i^{\mathrm{tx}} \tag{B.32}$$

where P_i^{rx}, P_i^{comp}, and P_i^{tx} are the mean power consumptions of the emulated device assigned to client i during data reception, computation, and data transmission, respectively.

The total estimated energy for assignment ac is

$$\widehat{E}_i^p(ac) = \widehat{E}_i^{D,p} + \widehat{E}_i^{C,p}(ac) + \widehat{E}_i^{U,p} \tag{B.33}$$

B.6.5 Cost vectors

Finally, after iterating over all feasible task capacities, the server obtains

$$\mathtt{T}_i = \left[\widehat{T}_i^p(ac) \mid ac \in \mathtt{AC}_i\right], \qquad \mathtt{E}_i = \left[\widehat{E}_i^p(ac) \mid ac \in \mathtt{AC}_i\right] \tag{B.34}$$

These vectors are used as performance estimates by the client selection approaches considered in this work. In *MetaCS-FL*, they support the generation of the initial solution and the evaluation of candidate schedules during the metaheuristic search.

B.7 Dataset Preparation

The experiments use Flower Datasets to construct the local datasets assigned to each FL client. The datasets are loaded via the following identifiers: `uoft-cs/cifar10` for CIFAR-10, `zalando-datasets/fashion_mnist` for Fashion-MNIST, and `dair-ai/emotion` for Emotion. For Emotion, the `unsplit` subset is used.

For each execution, the number of dataset partitions equals the number of clients:

$$\texttt{num\_partitions} = |\mathcal{C}| = \begin{cases} 50, & \text{for 50-client systems,} \\ 100, & \text{for 100-client systems.} \end{cases} \tag{B.35}$$

This ensures a one-to-one mapping between clients and dataset partitions.

Two data-distribution settings are considered: IID and non-IID. The same high-level IID/non-IID partitioning strategy is used for CIFAR-10, Fashion-MNIST, and Emotion, with one implementation difference: CIFAR-10 and Fashion-MNIST partition predefined training and testing splits, whereas Emotion is partitioned once and then split locally into training and testing subsets. More specifically, CIFAR-10 uses the `train` and `test` splits, Fashion-MNIST also uses the `train` and `test` splits, and Emotion uses the `train` split from the `unsplit` subset as the source data for local partitioning.

B.7.1 IID partitioning

In the IID setting, client datasets are generated using Flower Datasets' `IidPartitioner`. The selected dataset split is divided into $|\mathcal{C}|$ partitions, and each client receives one partition that approximates the global class distribution.

For CIFAR-10 and Fashion-MNIST, the `train` and `test` splits are both partitioned IID. For Emotion, the available data are partitioned IID, and each local partition is then split into local training and testing subsets using an 80/20 split. Before this local split, the examples in the client partition are randomly permuted.

B.7.2 Non-IID partitioning

In the non-IID setting, client datasets are generated using the `PathologicalPartitioner` strategy available in Flower Datasets. This creates label-skewed partitions by assigning each client examples from two classes. Partitioning is performed according to the label

field, using deterministic class assignment, shuffling, and seed 42.

For CIFAR-10 and Fashion-MNIST, both training and testing splits are partitioned using this pathological configuration. For Emotion, the pathological partitioner is applied to the available data, after which each local partition is split into local training and testing subsets using the same 80/20 procedure.

B.7.3 Dataset preprocessing

Dataset preprocessing is performed after each client loads its local federated partition. Since the experiments combine image and text datasets, the preprocessing procedure depends on the dataset–model tuple. Table 44 summarizes the preprocessing steps applied.

Table 44: Dataset preprocessing used by each dataset–model tuple.

Dataset	Model	Preprocessing procedure
CIFAR-10	`Custom_CNN_CIFAR-10`	Load RGB images from the `img` field; keep input shape $32\times32\times3$; scale pixel values by 1/255.
Fashion-MNIST	`Custom_CNN_FashionMNIST`	Load grayscale images from the `image` field; keep input shape $28 \times 28 \times 1$; scale pixel values by 1/255.
Emotion	`Custom_BiLSTM_Attention_Emotion`	Load text from the `text` field; perform local 80/20 train/test split; normalize text; tokenize with a shared tokenizer; pad sequences to length 50.

For CIFAR-10 and Fashion-MNIST, preprocessing consists of scaling image pixel values by 1/255, producing normalized floating-point inputs in the range [0,1]. The spatial dimensions are preserved.

For Emotion, text preprocessing expands contractions, lowercases the text, collapses repeated whitespace, removes empty or invalid texts, and converts each sentence to a padded sequence of token indices. A shared tokenizer is built across all federated partitions, using a vocabulary size of 50,000 and an out-of-vocabulary token. Sequences are padded to length 50 using post-padding. GloVe embeddings are disabled in the BiLSTM-Attention experiments, so the embedding layer is randomly initialized and trained jointly with the model.

The implementation builds the shared tokenizer once and stores it in the experiment output directory. File locking is used to avoid concurrent tokenizer construction when

multiple client processes start at the same time. After the tokenizer has been created, the remaining clients reuse the same tokenizer file, ensuring that all clients use a consistent vocabulary mapping.

B.7.4 Task budgets, profiling loads, and stopping criteria

Each scheduled task by the server corresponds to a dataset-dependent amount of local data: one image sample for CIFAR-10 and Fashion-MNIST, and ten text samples for Emotion. Therefore, the number of samples processed in a round depends on both the task budget and the dataset-specific number of samples represented by each task.

In each round, the server schedules a fixed task budget for training and testing. For CIFAR-10, the per-round budget is 20,000 training tasks and 10,000 testing tasks. For Fashion-MNIST, the per-round budget is 24,000 training tasks and 10,000 testing tasks. For Emotion, the per-round budget is 13,344 training tasks and 8,340 testing tasks.

These task budgets correspond to processing 40% of the training samples and all test samples in each round. For CIFAR-10 and Fashion-MNIST, each task represents one image sample. Therefore, the CIFAR-10 budget corresponds to 20,000 training samples and 10,000 test samples, while the Fashion-MNIST budget corresponds to 24,000 training samples and 10,000 test samples. For Emotion, each task represents ten text samples. Therefore, the Emotion budget corresponds to 133,440 training samples and 83,400 test samples, matching 40% of the 333,600 training samples and all 83,400 test samples.

Table 45: Dataset sizes, scheduled tasks, and processed samples per round.

Dataset	Dataset Size		Scheduled Tasks per Round		Samples per Task	Processed Samples per Round	
	Training Samples	Test Samples	Training Tasks	Testing Tasks		Training Samples	Test Samples
CIFAR-10	50,000	10,000	20,000	10,000	1	20,000	10,000
Fashion-MNIST	60,000	10,000	24,000	10,000	1	24,000	10,000
Emotion	333,600	83,400	13,344	8,340	10	133,440	83,400

Table 45 defines the total per-round task budgets used by the server-side client selection strategies, together with the corresponding number of processed samples for each dataset. For a selected client, the number of processed samples depends on both the number of assigned tasks and the dataset-specific `samples_per_task` value.

In the general case, each client could specify how much of its local data it is willing to contribute for training and testing. The experiments adopt a simplified setting in which the feasible task loads are derived automatically from the client's local data availability.

As a result, task assignment capacities remain client-specific and phase-specific, but are determined from the sizes of the client's local training and testing partitions.

For each client, the implementation computes separate task assignment capacities for training and testing. When the upper bound is set to `client_capacity`, the maximum feasible number of tasks is obtained by dividing the local number of samples by `samples_per_task`. The lower bound and step size are converted using the same sample-to-task scaling, with the step size lower-bounded by one task. The resulting capacity set includes the lower bound, the client-specific upper bound, and the intermediate capacities generated by the configured step. Therefore, the maximum feasible capacity is preserved even when it is not aligned with the step size. Thus, the client selection strategy can only assign task loads that are feasible for the corresponding client, phase, and local data.

Clients are lightly profiled upon joining the system using 200 training tasks and 200 testing tasks. This profiling allows the server to estimate client performance under different task loads before performing client selection. As described earlier, initially available clients are profiled synchronously before the first FL round, whereas late-joining clients are profiled asynchronously upon connection, allowing the FL execution to continue while new clients are characterized.

All reported experiments use a maximum of 100 communication rounds. The execution stops when either the maximum number of rounds is reached or the target testing accuracy associated with the dataset and data-distribution setting is achieved. Table 46 summarizes the target accuracies used for the IID and non-IID settings, as well as the per-round deadline τ adopted for each dataset. Following (7), the target test accuracies for CIFAR-10 and Fashion-MNIST are defined according to the data-distribution setting. For Emotion, the target accuracies are based on typical results reported in the literature (26, 121). The server-side random seed used by the selection strategies is 777, while the dataset partitioning seed used for the non-IID pathological partitioner is 42.

Table 46: Stopping criteria and per-round deadlines used in the experiments.

Dataset	Target Accuracy IID	Target Accuracy Non-IID	Per-Round Deadline τ (s)
CIFAR-10	0.75	0.45	1800
Fashion-MNIST	0.85	0.70	300
Emotion	0.90	0.80	1500

B.8 Model Architectures and Training Hyperparameters

To account for mobile and edge resource constraints, the experiments use lightweight neural network architectures. CIFAR-10 uses the CNN proposed by Nishio *et al.* (7), Fashion-MNIST uses a CNN based on McMahan *et al.* (1) with minor modifications, and Emotion uses the BiLSTM-Attention baseline described by Saravia *et al.* (26).

All models are implemented using Keras and trained using the Adam optimizer (35). The learning rate is fixed to 0.001. Adam uses $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$. In the reported experiments, all dataset–model tuples use sparse categorical cross-entropy and sparse categorical accuracy. Local model weights are aggregated on the server using FedAvg (1), which computes a weighted average proportional to each client's data size.

Table 47 summarizes the training hyperparameters used in the experiments. The CNN models used for CIFAR-10 and Fashion-MNIST are trained locally for five epochs with batch size 32, whereas the BiLSTM-Attention model used for Emotion is trained locally for four epochs with batch size 128. Evaluation uses batch size 32 for all datasets.

During training, batches are shuffled and no validation split is used. Keras derives the number of steps from the number of samples assigned to the client and the configured batch size. In the reported experiments, learning-rate scheduling callbacks are disabled, although the implementation supports optional learning-rate schedules and callbacks such as `ReduceLROnPlateau` and `EarlyStopping`.

Table 47: Training hyperparameters used by each dataset–model tuple.

Dataset	Model	Learning Rate	Local Epochs	Batch Size (Train)
CIFAR-10	`Custom_CNN_CIFAR-10`	0.001	5	32
Fashion-MNIST	`Custom_CNN_FashionMNIST`	0.001	5	32
Emotion	`Custom_BiLSTM_Attention_Emotion`	0.001	4	128

B.8.1 CIFAR-10 CNN

For CIFAR-10, the experiments use `Custom_CNN_CIFAR-10`, based on the CNN architecture adopted in FedCS (7). The input has shape $32 \times 32 \times 3$. The model contains six 3×3 convolutional layers organized into three blocks. Each convolution is followed by batch normalization and ReLU activation, and a 2×2 max-pooling layer is applied after every two convolutions. The classifier consists of a flattening layer, dense layers with 382 and 192 units, and a 10-unit softmax output layer. Table 48 summarizes this architecture.

Table 48: Architecture of the CIFAR-10 CNN model.

Stage	Layers
Input	$32 \times 32 \times 3$ image
Block 1	Conv2D(32) + BatchNorm + ReLU; Conv2D(32) + BatchNorm + ReLU; MaxPool(2×2)
Block 2	Conv2D(64) + BatchNorm + ReLU; Conv2D(64) + BatchNorm + ReLU; MaxPool(2×2)
Block 3	Conv2D(128) + BatchNorm + ReLU; Conv2D(128) + BatchNorm + ReLU; MaxPool(2×2)
Classifier	Flatten; Dense(382, ReLU); Dense(192, ReLU); Dense(10, softmax)

During local training, each selected client trains for 5 epochs with batch size 32.

B.8.2 Fashion-MNIST CNN

For Fashion-MNIST, the experiments use `Custom_CNN_FashionMNIST`. This model is based on the CNN model adopted by McMahan *et al.* (1), with minor modifications to improve computational efficiency. The input has shape $28 \times 28 \times 1$. The architecture contains two convolutional blocks. The first block applies a 3×3 convolution with 32 filters and ReLU activation, followed by 2×2 max pooling. The second block applies a 3×3 convolution with 128 filters and ReLU activation, followed by another max-pooling layer. The classifier contains a flattening layer, dropout with rate 0.5, and a 10-unit softmax output layer. Table 49 summarizes this architecture.

Table 49: Architecture of the Fashion-MNIST CNN model.

Stage	Layers
Input	$28 \times 28 \times 1$ image
Block 1	Conv2D(32, ReLU); MaxPool(2×2)
Block 2	Conv2D(128, ReLU); MaxPool(2×2)
Classifier	Flatten; Dropout(0.5); Dense(10, softmax)

During local training, each selected client trains for 5 epochs with batch size 32.

B.8.3 Emotion BiLSTM-Attention

For Emotion, the experiments use `Custom_BiLSTM_Attention_Emotion`, following the BiLSTM-Attention baseline described by Saravia *et al.* (26). The model receives padded token sequences of length 50. The first layer is an embedding layer with vocabulary size 50,000 and embedding dimension 100. Since GloVe embeddings are disabled in the reported configuration, this embedding layer is randomly initialized and trained together with the rest of the model.

The sequence representation is processed by a bidirectional LSTM layer with 64 units and `return_sequences = True`. An additive attention layer computes a sentence-level context vector from the BiLSTM hidden states. The classifier consists of a dense layer with 64 units, dropout with rate 0.5, and a 6-unit softmax output layer. Table 50 summarizes this architecture.

Table 50: Architecture of the Emotion BiLSTM-Attention model.

Stage	Layers
Input	Token sequence of length 50
Embedding	Embedding(50,000, 100), randomly initialized
Sequence encoder	Bidirectional LSTM(64, return sequences)
Attention	Additive attention over BiLSTM hidden states
Classifier	Dense(64, ReLU); Dropout(0.5); Dense(6, softmax)

During local training, each selected client trains for 4 epochs with batch size 128.

APPENDIX C — Theoretical Properties of MetaCS-FL

This appendix formalizes the key properties of MetaCS-FL that complement the framework definition and optimization model introduced in Chapter 5. Unless otherwise stated, the notation follows that chapter: $\mathcal{C}_r$ denotes the eligible client set, $\mathcal{S}_r$ the selected set, x_i the workload assigned to client i, t the total workload, τ the deadline, $\widetilde{\mathtt{AC}}_{i,r}$ the reliability-aware capacity set, rs_i the reliability score, and $F(\mathcal{Z}_r)$ the MetaCS-FL objective evaluated on a candidate schedule. Auxiliary notation introduced only in this appendix uses distinct symbols, such as $\mathcal{Z}_r$ for a generic candidate schedule and $\mathfrak{f}$ for the learning objective.

The appendix relies on standard concepts from finite combinatorial search spaces (113), weighted scalarization and Pareto optimality (112), entropy (152), stochastic expectation and Markov processes (153), and asymptotic cost analysis (62). Section C.1 establishes the feasible schedule space, weighted objective, class-distribution and utility scores, and best-solution monotonicity. Section C.2 positions the diversity term as a soft fairness regularizer. Section C.3 derives reliability-aware scheduling properties, including bounded reliability, smoothed reliability behavior, workload restriction, expected workload loss, and feasibility recovery. Finally, Section C.4 formalizes the selection-cost reduction obtained through event-driven reuse.

C.1 Client-Task Scheduling and Objective Properties

This section formalizes the optimization structure induced by MetaCS-FL. It first shows that, although the feasible schedule space is finite under finite client and capacity sets, the number of possible schedules can be extremely large. It then casts the weighted objective as a scalarization of multiple scheduling criteria and establishes the monotonicity property of the best-so-far solution maintained during metaheuristic refinement.

C.1.1 Feasible Schedule Space

For a fixed round r, consider a generic candidate schedule

$$\mathcal{Z}_r = (z_{1,r}, z_{2,r}, \ldots, z_{n,r})$$

where $z_{i,r}$ denotes the number of tasks assigned to client i by the candidate schedule $\mathcal{Z}_r$. A feasible schedule must satisfy

$$z_{i,r} \in \widetilde{\mathtt{AC}}_{i,r}, \quad \forall i \in \mathcal{C}_r, \qquad \sum_{i \in \mathcal{C}_r} z_{i,r} = t$$

and, when a deadline is defined,

$$\mathrm{M}_r(\mathcal{Z}_r) \leq \tau$$

Proposition 1 (Finite feasible schedule space)**.** *For a fixed round r, if $\mathcal{C}_r$ is finite and each $\widetilde{\mathtt{AC}}_{i,r}$ is finite, then the feasible schedule space of MetaCS-FL is finite.*

Proof. Each client can only receive values from its finite capacity set. Thus, all possible schedules are contained in the finite Cartesian product

$$\prod_{i \in \mathcal{C}_r} \widetilde{\mathtt{AC}}_{i,r}$$

The workload and deadline constraints only select a subset of this finite set. Therefore, the feasible schedule space is finite. □

Finiteness does not imply tractability. If $n = 100$, $t = 20{,}000$, and each client can receive any value in $\{0, \ldots, 500\}$, the number of assignments satisfying only the workload and upper-bound constraints is the bounded weak-composition count (113)

$$\sum_{k=0}^{\left\lfloor \frac{t}{501} \right\rfloor} (-1)^k \cdot \binom{n}{k} \cdot \binom{t - 501 \cdot k + n - 1}{n - 1}$$

This search space is already prohibitively large under only the workload constraint and per-client capacity bounds. Related FL scheduling methods, including *OLAR* (8), *Fed-LBAP/MinCost* (99), *MEC/ECMTC* (28), and *(MC)²MKP* (60), similarly treat client selection or workload assignment as a constrained scheduling problem. Exhaustively evaluating such schedules according to time, energy, reliability, utility, class-distribution, and diversity criteria is impractical, motivating a structured initial solution followed by metaheuristic refinement.

C.1.2 Pareto Interpretation of the Weighted Objective

The MetaCS-FL objective minimizes makespan and energy while maximizing diversity, class-distribution quality, and utility. The makespan and energy components are aligned with scheduling-oriented FL formulations that optimize execution time, energy consumption, or both, including *OLAR* (8), *MEC/ECMTC* (28), and *(MC)²MKP* (60). The class-distribution component is related to methods that account for non-IID data, class imbalance, or distribution quality, including *Fed-LBAP/MinCost* (99), *FedSampling* (78), *FedMCCS* (5), *FedCAB* (17), and *FLICS-OPT* (21).

Unlike these methods, MetaCS-FL explicitly incorporates class coverage and class imbalance into the scheduling objective through K_r, allowing data-distribution quality to be traded against time, energy, diversity, and utility. The utility component is conceptually related to learning-aware client selection methods such as *Oort* (46), which balances statistical utility and execution efficiency, and *FedPNS* (9), which adjusts node selection probabilities according to their contribution to convergence.

Following standard multi-objective optimization practice, MetaCS-FL uses weighted scalarization to obtain a scalar optimization criterion from the individual criteria (112). Since this criterion is written in minimization form, criteria that are naturally maximized are included with negative signs:

$$\Omega_1 = \mathrm{M}_r, \quad \Omega_2 = \Sigma_r, \quad \Omega_3 = -\mathrm{D}_r, \quad \Omega_4 = -\mathrm{K}_r, \quad \Omega_5 = -\mathrm{U}_r$$

The scalarized objective can then be written as

$$F(\mathcal{Z}_r) = \sum_{j=1}^{5} w_j \cdot \Omega_j(\mathcal{Z}_r)$$

Proposition 2 (Pareto optimality of exact weighted-sum minimizers)**.** *Let* $J^+ = \{j : w_j > 0\}$. *If* $\mathcal{Z}_r^\star$ *globally minimizes* F *over the feasible schedules, then* $\mathcal{Z}_r^\star$ *is Pareto optimal with respect to the positively weighted objectives* $(\Omega_j)_{j\in J^+}$.

Proof. If $\mathcal{Z}_r^\star$ were not Pareto optimal with respect to $(\Omega_j)_{j\in J^+}$, another feasible schedule $\mathcal{Z}_r'$ would satisfy $\Omega_j(\mathcal{Z}_r') \leq \Omega_j(\mathcal{Z}_r^\star)$ for all $j \in J^+$, with at least one strict inequality. Since $w_j > 0$ for all $j \in J^+$ and $w_j = 0$ for $j \notin J^+$, this would imply $F(\mathcal{Z}_r') < F(\mathcal{Z}_r^\star)$, contradicting global optimality. □

This property characterizes the exact weighted-sum problem and does not guarantee

global optimality of the metaheuristic output. In MetaCS-FL, the scalarized objective defines a Pareto-consistent preference over positively weighted criteria, while the metaheuristic uses this objective to guide the search toward a high-quality feasible schedule.

C.1.3 Class-Distribution and Utility Score Properties

The class-distribution score K_r and utility score U_r encode learning-aware aspects of the schedule. While makespan and energy capture system cost, these two terms encourage the selected workload to be informative from the data and model-performance perspectives.

Proposition 3 (Bounded class-distribution score)**.** *Assume that the assigned class counts satisfy* $0 \leq \mathrm{K}_{\mathcal{S}}[k] \leq \mathrm{K}_{\mathcal{C}}[k]$ *for every* $k \in \Gamma$*, with* $\mathrm{K}_{\mathcal{C}}[k] > 0$*, and that the assigned workload satisfies* $\sum_{k\in\Gamma} \mathrm{K}_{\mathcal{S}}[k] = t$*. If* $\beta \in [0,1]$*, then*

$$0 \leq \mathrm{K}_r \leq 1$$

Proof. For each class $k \in \Gamma$, the ratio

$$\frac{\mathrm{K}_{\mathcal{S}}[k]}{\mathrm{K}_{\mathcal{C}}[k]}$$

belongs to [0,1]. Therefore, the class coverage term

$$\mathrm{K}_{Cov} = \frac{1}{m} \cdot \sum_{k\in\Gamma} \frac{\mathrm{K}_{\mathcal{S}}[k]}{\mathrm{K}_{\mathcal{C}}[k]}$$

also belongs to [0,1].

The imbalance term K_{Imb} is non-negative because it is defined from a square root of squared deviations. It is equal to 0 when the assigned workload is evenly distributed across the m classes, that is, when $\mathrm{K}_{\mathcal{S}}[k] = t/m$ for all $k \in \Gamma$. Under a non-negative assigned class distribution with total workload t, the normalized deviation is at most 1, so $1 - \mathrm{K}_{Imb} \in [0,1]$. Since

$$\mathrm{K}_r = \beta \cdot \mathrm{K}_{Cov} + (1 - \beta) \cdot (1 - \mathrm{K}_{Imb})$$

is a convex combination of two values in [0,1], it follows that $0 \leq \mathrm{K}_r \leq 1$. □

This property shows that K_r behaves as a normalized score. Higher values correspond to schedules that use a larger fraction of the available class-specific data while keeping the assigned workload more balanced across classes.

Proposition 4 (Bounded utility score)**.** *Assume that the normalized training and test losses used to compute ϕ_i and ψ_i belong to* [0,1]*, and let $\alpha \in [0,1]$. If the schedule satisfies $\sum_{i\in\mathcal{C}_r} x_i = t$, then*

$$0 \le \mathrm{U}_r \le 1$$

Proof. Since the normalized losses belong to [0,1], the terms ϕ_i and ψ_i also belong to [0,1]. Therefore,

$$\alpha \cdot \phi_i + (1-\alpha) \cdot \psi_i \in [0,1]$$

because it is a convex combination. For each selected client,

$$u_i = \frac{x_i}{t} \cdot (\alpha \cdot \phi_i + (1-\alpha) \cdot \psi_i)$$

satisfies

$$0 \le u_i \le \frac{x_i}{t}$$

Summing over all clients gives

$$0 \le \mathrm{U}_r = \sum_{i\in\mathcal{C}_r} u_i \le \sum_{i\in\mathcal{C}_r} \frac{x_i}{t} = 1$$

□

Thus, U_r can be interpreted as a workload-weighted learning utility score. A selected client contributes more to U_r when it receives more workload and exhibits stronger historical learning or generalization behavior.

Proposition 5 (Objective preference for improved class-distribution and utility scores)**.** *Consider two feasible schedules $\mathcal{Z}_a$ and $\mathcal{Z}_b$ with equal makespan, energy, and diversity. If*

$$\mathrm{K}_r(\mathcal{Z}_a) \ge \mathrm{K}_r(\mathcal{Z}_b), \qquad \mathrm{U}_r(\mathcal{Z}_a) \ge \mathrm{U}_r(\mathcal{Z}_b)$$

with at least one strict inequality, and if the strictly improved score has positive weight, then

$$F(\mathcal{Z}_a) < F(\mathcal{Z}_b)$$

Proof. Since the schedules have equal makespan, energy, and diversity, the objective difference depends only on the class-distribution and utility terms:

$$F(\mathcal{Z}_a) - F(\mathcal{Z}_b) = -w_4 \cdot (\mathrm{K}_r(\mathcal{Z}_a) - \mathrm{K}_r(\mathcal{Z}_b)) - w_5 \cdot (\mathrm{U}_r(\mathcal{Z}_a) - \mathrm{U}_r(\mathcal{Z}_b))$$

The right-hand side is non-positive because w_4,$w_5 \ge 0$ and both score differences are

non-negative. If at least one improved term has a positive weight, the right-hand side is strictly negative. Hence, $F(\mathcal{Z}_a) < F(\mathcal{Z}_b)$. □

This property states that, all else being equal, MetaCS-FL prefers schedules with better class-distribution quality or higher learning utility whenever the corresponding objective weights are positive.

C.1.4 Best-Solution Monotonicity

Let $\mathcal{Z}_k^{\text{best}}$ be the best solution stored after iteration k of the employed metaheuristic.

Proposition 6 (Best-so-far objective monotonicity)**.** *If MetaCS-FL replaces* $\mathcal{Z}_k^{\text{best}}$ *only when a candidate has a lower objective value, then*

$$F(\mathcal{Z}_{k+1}^{\text{best}}) \leq F(\mathcal{Z}_k^{\text{best}})$$

Proof. At iteration $k+1$, either the candidate improves the current best solution, in which case the stored objective decreases, or it does not, in which case the stored solution remains unchanged. Therefore, the best-so-far objective value cannot increase. □

This property does not depend on the specific metaheuristic. It only requires that the search keeps a best-so-far solution and updates it when a lower objective value is found. Therefore, it establishes the monotonicity of the stored best objective value, not the convergence of the search process.

C.2 Diversity as a Fairness Regularizer

This section describes the role of the diversity term in the MetaCS-FL objective. The diversity score acts as a history-aware regularizer that favors schedules with more balanced recent participation among eligible clients. The properties below show that the score is bounded, is maximized by uniform participation, and induces an objective preference for more balanced schedules when the remaining criteria are unchanged.

The diversity score D_r is defined in Chapter 5 from the entropy contributions d_i. For each client $i \in \mathcal{C}_r$, ρ_i denotes the number of recent participations of client i in the considered window, including the candidate participation in round r. Thus, ρ_i depends on the candidate schedule being evaluated: selected clients receive one additional participation

in the current window, while non-selected clients do not. The entropy contribution is

$$d_i = -\frac{\rho_i}{\sum_{j\in\mathcal{C}_r}\rho_j}\cdot\log_e\left(\frac{\rho_i}{\sum_{j\in\mathcal{C}_r}\rho_j}\right)$$

for $\rho_i \neq 0$, and $d_i = 0$ otherwise. The diversity score is

$$\mathrm{D}_r = \frac{\sum_{i\in\mathcal{C}_r} d_i}{\log_e(|\mathcal{C}_r|)}$$

Thus, D_r is the Shannon evenness index (111) of the recent participation counts among the eligible clients. This term is related to diversity-aware FL methods such as *DivFL* (15), which promotes representative client subsets in gradient space, and to fairness-oriented methods such as *q-FedAvg* (95), which improves the uniformity of client performance through aggregation reweighting. MetaCS-FL differs from these approaches by using recent participation entropy as a scheduling regularizer.

Proposition 7 (Bounds and maximum of the diversity score)**.** *If $|\mathcal{C}_r| > 1$ and at least one eligible client has participated in the considered window, then*

$$0 \leq \mathrm{D}_r \leq 1$$

The maximum value is reached when participation is uniform across the eligible clients.

Proof. Let

$$p_i = \frac{\rho_i}{\sum_{j\in\mathcal{C}_r}\rho_j}$$

Then $\sum_{i\in\mathcal{C}_r} d_i$ is the Shannon entropy of the participation distribution $(p_i)_{i\in\mathcal{C}_r}$, with zero-count clients contributing 0. Shannon entropy over $|\mathcal{C}_r|$ elements is bounded between 0 and $\log_e(|\mathcal{C}_r|)$, and the upper bound is attained when $p_i = 1/|\mathcal{C}_r|$ for all $i \in \mathcal{C}_r$. After normalization by $\log_e(|\mathcal{C}_r|)$, these entropy bounds become $0 \leq \mathrm{D}_r \leq 1$. □

Proposition 8 (Implicit fairness pressure)**.** *Consider two feasible schedules $\mathcal{Z}_a$ and $\mathcal{Z}_b$ with equal makespan, energy, class-distribution score, and utility. If*

$$\mathrm{D}_r(\mathcal{Z}_a) > \mathrm{D}_r(\mathcal{Z}_b)$$

and $w_3 > 0$, then

$$F(\mathcal{Z}_a) < F(\mathcal{Z}_b)$$

Proof. Since the schedules differ only in diversity,

$$F(\mathcal{Z}_a) - F(\mathcal{Z}_b) = -w_3\cdot(\mathrm{D}_r(\mathcal{Z}_a) - \mathrm{D}_r(\mathcal{Z}_b))$$

The right-hand side is negative because $w_3 > 0$ and $\mathrm{D}_r(\mathcal{Z}_a) > \mathrm{D}_r(\mathcal{Z}_b)$. Therefore, $F(\mathcal{Z}_a) < F(\mathcal{Z}_b)$. □

Thus, D_r penalizes concentration in participation by assigning a higher objective value to schedules that repeatedly favor the same clients, all else being equal. It does not impose round-robin selection or guarantee that every client is selected within a certain number of rounds. Instead, it acts as a soft fairness pressure, encouraging broader participation unless the loss in diversity is offset by gains in time, energy, class distribution, or utility.

C.3 Reliability-Aware Scheduling

MetaCS-FL employs reliability-aware scheduling to account for clients that are eligible before selection but may fail to complete their assigned workload afterward. This scheduling strategy combines a reliability score with a reliability-aware restriction of each client's admissible capacity set, so that clients with weaker availability or completion history remain eligible, but their assignable workload is reduced. It is related to availability-aware selection methods such as *RIFLES* (22), which uses heartbeat-based availability forecasting for client scheduling, as well as *FedAWE* (20) and *CA-Fed* (18), which address FL under heterogeneous client availability. The properties below show how the reliability score remains bounded, tracks client success behavior, limits workload exposure for unreliable clients, and preserves feasibility through finite relaxation when necessary.

We consider clients whose availability evolves over time. Let $A_{i,r} \in \{0,1\}$ denote the availability state of client i in round r, where $A_{i,r} = 1$ indicates that the client is available and $A_{i,r} = 0$ indicates that it is unavailable. To capture temporal persistence in availability, we use the following two-state Markov process:

$$\Pr(A_{i,r} = 0 \mid A_{i,r-1} = 1) = p_i^{\mathrm{off}}, \qquad \Pr(A_{i,r} = 1 \mid A_{i,r-1} = 0) = p_i^{\mathrm{on}}$$

Thus, clients may remain in the same availability state across consecutive rounds. This model is more expressive than an independent per-round availability model since the current state depends on the previous one. At the same time, it remains simple to parameterize and reproduce. Markovian availability has also been considered in availability-aware FL; for example, *CA-Fed* (18) models client availability using a finite-state Markov chain. Other works adopt probability-based availability assumptions, such as *FedAWE* (20), or rely on availability prediction from historical signals, as in *RIFLES* (22). Therefore, the two-state Markov process provides a controlled middle ground between independent avail-

ability assumptions and more complex availability-aware or forecasting-based models.

In this model, pre-selection unavailability changes the eligible set $\mathcal{C}_r$, while post-selection completion failure changes the subset of selected clients that actually return updates. The model distinguishes between these two events because they affect the system differently. Pre-selection unavailability affects candidate-set construction: a client unavailable before selection is excluded from $\mathcal{C}_r$ and receives the availability penalty λ_{avail}. Post-selection completion failure affects the execution outcome: a selected client that does not complete its processing is excluded from the completed-client subset and receives the completion penalty λ_{compl}. Since completion failures consume a selection slot and may directly affect the current round, λ_{compl} is configured to be at least as large as λ_{avail}.

C.3.1 Reliability Score Properties

The following properties use standard notions from probability and stochastic processes, including expected value, Bernoulli indicators, and Markovian state evolution (153).

Let $rs_{i,r} \in [0,1]$ denote the reliability score of client i in round r. The score is updated from the previous reliability score and from the reliability observation obtained in the preceding round:

$$rs_{i,r} = \mu \cdot rs_{i,r-1} + (1-\mu) \cdot (1-\lambda_{i,r-1})$$

where $\lambda_{i,r-1} \in [0,1]$ is the penalty assigned to client i based on its behavior in round $r-1$, and $\mu \in [0,1]$ controls how much past behavior is preserved. Equivalently, if

$$\zeta_{i,r-1} \coloneqq 1 - \lambda_{i,r-1}$$

then $\zeta_{i,r-1}$ is the reliability observation obtained from the previous round. Values of $\zeta_{i,r-1}$ close to 1 indicate successful or non-penalized behavior, whereas smaller values indicate availability or completion problems.

The reliability score is therefore an exponentially weighted average of past reliability observations. The parameter μ determines the memory of this average. When μ is small, the score gives more importance to the most recent observation. When μ is large, the score changes more slowly and preserves more information from previous rounds.

Proposition 9 (Bounded reliability score)**.** *If* $rs_{i,r-1} \in [0,1]$, $\mu \in [0,1]$, *and* $\lambda_{i,r-1} \in [0,1]$, *then* $rs_{i,r} \in [0,1]$.

Proof. Since $\lambda_{i,r-1} \in [0,1]$, the observation $1 - \lambda_{i,r-1}$ also belongs to [0,1]. The update

$$rs_{i,r} = \mu \cdot rs_{i,r-1} + (1 - \mu) \cdot (1 - \lambda_{i,r-1})$$

is therefore a convex combination of two values in [0,1]. Hence, $rs_{i,r} \in [0,1]$. □

Let $\mathbb{E}[\cdot]$ denote the expected value, or average value, over the randomness of client behavior. In this context, the randomness may come from intermittent availability, temporary disconnections, or completion failures. Thus, $\mathbb{E}[\zeta_{i,r}]$ represents the average reliability observation that would be obtained for client i in round r over repeated executions of the same stochastic process.

A sequence of reliability observations $(\zeta_{i,r})_{r\geq 0}$ has a stationary mean if its average behavior does not change over time. In particular,

$$\mathbb{E}[\zeta_{i,r}] = \theta_i, \qquad \text{for all } r$$

A stationary mean does not imply that the client behaves identically in every round. Individual observations may still vary from round to round. Rather, it indicates that the client's long-run average reliability remains stable over time. For example, a client whose long-run availability and completion behavior remains statistically unchanged can be regarded as producing reliability observations with a stationary mean.

Proposition 10 (Reliability as a smoothed success measure)**.** *Let $\zeta_{i,r} = 1 - \lambda_{i,r}$ be the reliability observation of client i in round r. Suppose that the sequence $(\zeta_{i,r})_{r\geq 0}$ has stationary mean $\mathbb{E}[\zeta_{i,r}] = \theta_i$, and let $0 \leq \mu < 1$. Then repeated application of the reliability update yields*

$$\lim_{r\to\infty} \mathbb{E}[rs_{i,r}] = \theta_i$$

Proof. Using $\zeta_{i,r-1} = 1 - \lambda_{i,r-1}$, the reliability update can be written as

$$rs_{i,r} = \mu \cdot rs_{i,r-1} + (1 - \mu) \cdot \zeta_{i,r-1}$$

Applying the expectation operator $\mathbb{E}[\cdot]$ to both sides gives

$$\mathbb{E}[rs_{i,r}] = \mathbb{E}[\mu \cdot rs_{i,r-1} + (1 - \mu) \cdot \zeta_{i,r-1}]$$

By the linearity of expectation, and since μ is a constant, this becomes

$$\mathbb{E}[rs_{i,r}] = \mu \cdot \mathbb{E}[rs_{i,r-1}] + (1 - \mu) \cdot \mathbb{E}[\zeta_{i,r-1}]$$

Since the observation sequence has a stationary mean θ_i, the expected reliability observation is the same in every round. Hence,

$$\mathbb{E}[\zeta_{i,r-1}] = \theta_i$$

Substituting this into the previous expression gives

$$\mathbb{E}[rs_{i,r}] = \mu \cdot \mathbb{E}[rs_{i,r-1}] + (1-\mu) \cdot \theta_i$$

This recursive relation shows that the expected reliability score in round r depends on the expected score in the previous round and on the client's long-run mean reliability observation. Expanding this relation over successive rounds gives

$$\mathbb{E}[rs_{i,r}] = \mu^r \cdot rs_{i,0} + (1-\mu^r) \cdot \theta_i$$

where $rs_{i,0}$ is the initial reliability score. Since $0 \leq \mu < 1$, the term μ^r decreases to zero as r grows. Therefore, the influence of the initial score disappears asymptotically, and

$$\lim_{r\to\infty} \mathbb{E}[rs_{i,r}] = \theta_i$$

□

The proposition shows that, when the client's reliability observations have a stable long-run average, the expected reliability score converges to that average. This statement is not specific to the two-state Markov availability model; it only requires $\mathbb{E}[\zeta_{i,r}] = \theta_i$ for all rounds. Therefore, $rs_{i,r}$ can be interpreted as a smoothed estimate of the client's mean reliability behavior, rather than as a score that depends only on the most recent round.

This convergence is in expectation, not necessarily along individual executions. Under non-stationary behavior, such as when a client becomes more unstable over time or recovers after poor connectivity, the update should be interpreted as an adaptive smoothing mechanism rather than as a convergence guarantee. In this case, μ controls the adaptation rate: smaller values make the score more responsive to recent penalties, while larger values preserve more history and smooth temporary availability or completion fluctuations.

C.3.2 Reliability-Aware Capacity Restriction

Let $x_i^{\max}(s)$ be the largest workload admissible for client i when its reliability score is s, with $s \in [0,1]$.

Proposition 11 (Monotonic workload exposure)**.** *If $s_1 \leq s_2$, then*

$$x_i^{\max}(s_1) \leq x_i^{\max}(s_2)$$

Proof. The number of admissible capacity slots under the reliability score s is

$$\max\left(1, \lfloor s \cdot |\mathtt{AC}_i| \rfloor\right)$$

which is non-decreasing in s. Since $\mathtt{AC}_i$ is ordered, allowing more capacity slots cannot reduce the largest admissible workload. Therefore, if $s_1 \leq s_2$, then

$$x_i^{\max}(s_1) \leq x_i^{\max}(s_2)$$

□

This property ensures that less reliable clients are not excluded from selection, but remain eligible with a reduced maximum assignable workload.

C.3.3 Expected Workload Loss

Let $\chi_{i,r}^{\text{fail}}$ denote the probability that the selected client i fails to complete its assigned workload in round r, and let $\xi_{i,r} \in \{0,1\}$ be the corresponding Bernoulli indicator, where $\xi_{i,r} = 1$ indicates a completion failure. The failed workload in round r is defined as

$$\mathfrak{Q}_r^{\text{fail}} = \sum_{i \in \mathcal{S}_r} \xi_{i,r} \cdot x_i$$

Since $\mathbb{E}[\xi_{i,r}] = \chi_{i,r}^{\text{fail}}$ and the assigned workload x_i is fixed after scheduling, the expected failed workload is

$$\mathbb{E}[\mathfrak{Q}_r^{\text{fail}}] = \mathbb{E}\left[\sum_{i \in \mathcal{S}_r} \xi_{i,r} \cdot x_i\right] = \sum_{i \in \mathcal{S}_r} x_i \cdot \mathbb{E}[\xi_{i,r}] = \sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i$$

This quantity represents the expected workload loss in round r, measured by the number of assigned tasks that are not completed because selected clients fail after selection.

Proposition 12 (Reliability-aware upper bound on expected failed workload)**.** *Under any feasible MetaCS-FL schedule,*

$$\mathbb{E}[\mathfrak{Q}_r^{\text{fail}}] \leq \sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i^{\max}(r s_i)$$

Proof. For every selected client, feasibility implies

$$x_i \leq x_i^{\max}(rs_i)$$

Since $\chi_{i,r}^{\text{fail}} \geq 0$, multiplying both sides by $\chi_{i,r}^{\text{fail}}$ preserves the inequality:

$$\chi_{i,r}^{\text{fail}} \cdot x_i \leq \chi_{i,r}^{\text{fail}} \cdot x_i^{\max}(rs_i)$$

Summing over all selected clients gives

$$\sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i \leq \sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i^{\max}(rs_i)$$

Since

$$\mathbb{E}[\mathfrak{Q}_r^{\text{fail}}] = \sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i$$

we obtain

$$\mathbb{E}[\mathfrak{Q}_r^{\text{fail}}] \leq \sum_{i \in \mathcal{S}_r} \chi_{i,r}^{\text{fail}} \cdot x_i^{\max}(rs_i)$$

which proves the bound. □

This bound formalizes the purpose of reliability-aware capacity restriction: by reducing $x_i^{\max}(rs_i)$ for less reliable clients, MetaCS-FL reduces the maximum workload at risk of being lost if they fail to complete. Moreover, reallocating $\Delta > 0$ tasks from client j to client i does not increase expected failed workload whenever

$$\chi_{i,r}^{\text{fail}} \leq \chi_{j,r}^{\text{fail}}$$

Indeed, the expected change is

$$\Delta \cdot \chi_{i,r}^{\text{fail}} - \Delta \cdot \chi_{j,r}^{\text{fail}} \leq 0$$

The probability $\chi_{i,r}^{\text{fail}}$ may be fixed or may vary with the assigned workload, as in the intermittent-availability simulation (see Section B.5). In the latter case, assigning fewer tasks can reduce not only the workload lost if a client fails but also the likelihood of that failure occurring. Therefore, reliability-aware scheduling can reduce failure exposure in two complementary ways: by limiting the workload lost if a client fails and, when failures are workload-dependent, by also reducing failure likelihood.

C.3.4 Finite Feasibility Recovery

Reliability-aware restriction can make the workload infeasible by removing capacity slots from less reliable clients. However, if the nominal capacity sets are sufficient to assign the full workload, feasibility can be recovered by gradually restoring removed slots.

Proposition 13 (Finite feasibility recovery)**.** *Assume that the nominal capacity sets* $\{\mathtt{AC}_i\}_{i\in\mathcal{C}_r}$ *contain at least one feasible schedule satisfying* $\sum_{i\in\mathcal{C}_r} x_i = t$*. If the restricted sets* $\{\widetilde{\mathtt{AC}}_{i,r}\}_{i\in\mathcal{C}_r}$ *are insufficient, then a relaxation procedure that restores valid nominal slots recovers feasibility after finitely many additions.*

Proof. Only finitely many capacity slots can be removed because there are finitely many clients and each capacity set is finite. In the worst case, all removed slots are restored, yielding the nominal capacity sets. Since the nominal problem is feasible by assumption, feasibility is recovered after finitely many restorations. □

C.4 Selection Overhead

This section connects the scheduling mechanism to computational overhead by formalizing how event-driven reuse reduces selection costs by avoiding unnecessary optimization calls in rounds where the previous schedule can be reused.

C.4.1 Event-Driven Selection Cost

Let R be the total number of FL rounds, and let B_R be the number of rounds in which MetaCS-FL triggers a new client selection. As in the computational-complexity analysis, let C_{init} denote the cost of generating the initial solution, I_{MH} the number of metaheuristic iterations, and C_{eval} the cost of evaluating $F(\mathcal{X})$ for one candidate solution. Let C_{reuse} denote the cost of retrieving a previously stored selection and its task assignments.

When a new selection is triggered, the selection cost is

$$C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}$$

When no new selection is needed, the framework reuses the latest selected clients and task assignments at cost C_{reuse}. Therefore, the total client-selection cost over R rounds is

$$T_{\text{cs}} = B_R \cdot (C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}) + (R - B_R) \cdot C_{\text{reuse}}$$

If the stored selection can be retrieved in constant time, then $C_{\text{reuse}} = \mathcal{O}(1)$, and

$$T_{\text{cs}} = \mathcal{O}\left(B_R \cdot (C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}) + (R - B_R)\right)$$

This distinction between reselection and reuse complements scheduling-oriented FL methods that account for the cost of making selection decisions under resource and availability constraints (7, 8, 22).

Proposition 14 (Selection-cost reduction by event-driven reuse)**.** *Compared with always generating a new selection, event-driven reuse reduces the total selection cost whenever*

$$B_R < R \quad \textit{and} \quad C_{\text{reuse}} < C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}$$

Proof. If a new selection is generated in every round, the total cost is

$$R \cdot (C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}})$$

With event-driven reuse, the total cost is

$$B_R \cdot (C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}) + (R - B_R) \cdot C_{\text{reuse}}$$

The cost reduction is therefore

$$(R - B_R) \cdot (C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}} - C_{\text{reuse}})$$

This quantity is positive whenever $B_R < R$ and

$$C_{\text{reuse}} < C_{\text{init}} + I_{\text{MH}} \cdot C_{\text{eval}}$$

Hence, event-driven reuse reduces the total selection cost under the stated conditions. □