

UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS MAROTTI PEREIRA

Optimizing Global Federated Learning: A Serverless Hierarchical Approach with Region-Aware Placement

NITERÓI

2026

UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS MAROTTI PEREIRA

Optimizing Global Federated Learning: A Serverless Hierarchical Approach with Region-Aware Placement

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science.
Topic Area: Computer Science.

Advisor:

LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND

NITERÓI

2026

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

P4360 Pereira, Matheus Marotti
Optimizing Global Federated Learning : A Serverless
Hierarchical Approach with Region-Aware Placement / Matheus
Marotti Pereira. - 2026.
98 f.

Orientador: Lúcia Maria de Assumpção Drummond.
Dissertação (mestrado)-Universidade Federal Fluminense,
Instituto de Computação, Niterói, 2026.

1. Aprendizado de máquina. 2. Computação paralela e
distribuída. 3. Computação em nuvem. 4. Produção
intelectual. I. Drummond, Lúcia Maria de Assumpção,
orientadora. II. Universidade Federal Fluminense. Instituto de
Computação. III. Título.

CDD - XXX

MATHEUS MAROTTI PEREIRA

Optimizing Global Federated Learning: A Serverless Hierarchical Approach with
Region-Aware Placement

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science.
Topic Area: Computer Science.

Approved on August 06th, 2026 by:

Prof. Lúcia M. A. Drummond, D.Sc. / IC-UFF (President)

Prof. Flavia Cristina Bernardini, D.Sc. / IC-UFF

Prof. Claude Martin Tadonki, D.Sc. / MINES ParisTech-PSL / CRI

Niterói
2026

*“You should enjoy the little detours to the fullest.
Because that’s where you’ll find the things
more important than what you want.”*

Yoshihiro Togashi

Acknowledgements

I am deeply grateful to my advisor, Lúcia Drummond, for the support and guidance she offered at every stage of this work, and for the opportunity to learn from her.

To my family, and above all to my parents, Paulo and Paula, and to my brother, Thiago, for the unconditional support and the affection that carried me here. Nothing in this work would have been possible without them.

To Daniela, for the love and the companionship, and for being present through every part of this journey, including the difficult ones.

To my friends, and in particular to Luan and Thiago, who went through the undergraduate years alongside me and are now with me in the graduate program as well. Sharing this path with them made it lighter. To André, Carlos, Guilherme, and Luiz Gabriel, for the friendship that endured across all these years.

To my laboratory colleagues, Alan, Bernardo, Daniel, Douglas, Lucas, and Miguel, for the exchanges, the shared work, the countless coffees, and the everyday company that makes research a collective endeavour rather than a solitary one.

To my cats, Mono and Jorel. Jorel kept me company through this work and did not live to see it finished. He is missed on every page.

This research was supported by scholarships from CAPES and Petrobras, which funded my studies throughout the master's program.

The experiments reported in this dissertation were made possible by the Google Cloud research credits program, whose support made it possible to evaluate the proposed architecture on real cloud infrastructure.

Resumo

A adoção em larga escala de modelos de *Machine Learning* (ML) enfrenta crescentes desafios de escalabilidade e privacidade, impulsionados pelo aumento do volume de dados gerados e por rigorosas regulamentações de proteção de dados. O Aprendizado Federado (FL) mitiga esses riscos ao manter os dados nos dispositivos locais e transmitir apenas os pesos dos modelos. No entanto, escalar o FL para clientes distribuídos globalmente introduz latência de comunicação e gargalos em redes transcontinentais. Embora o Aprendizado Federado Hierárquico (HFL) reduza essa carga por meio de agregações intermediárias na borda (*edge*), sua implantação em ambientes tradicionais de Computação em Nuvem normalmente depende de máquinas virtuais persistentes (*always-on*) de Infraestrutura como Serviço (IaaS). Isso cria um gargalo operacional: a natureza intermitente dos ciclos de treinamento do FL deixa esses recursos provisionados ociosos por longos períodos, resultando em altos custos de infraestrutura e sobrecarga de gerenciamento. Para solucionar essas ineficiências, esta dissertação utiliza o paradigma de Computação *Serverless* para propor uma arquitetura de HFL *Serverless*. Ao abstrair o gerenciamento subjacente dos servidores e executar tarefas em contêineres efêmeros e acionados por eventos, essa abordagem atinge um modelo de execução totalmente baseado no uso (*pay-per-use*) que escala nativamente para zero. Nossa arquitetura é orquestrada por meio de *workflows* gerenciados em nuvem, eliminando os custos de infraestrutura ociosa ao mesmo tempo em que reduz a carga de gerenciamento da orquestração persistente. Além disso, introduzimos um método de otimização sensível à região (*region-aware*) para alocar estrategicamente o servidor global de parâmetros, ponderando a latência de rede intercontinental, os custos de transferência de saída de dados (*egress costs*) e as tarifas regionais de computação. A avaliação empírica foi conduzida por meio de implantações reais no Google Cloud Platform (GCP) utilizando *benchmarks* padrão (MNIST, CIFAR-10) e um estudo de caso geoespacial em escala global (So2Sat) distribuído em quatro continentes. Os resultados demonstram que nossa abordagem hierárquica *serverless* reduz os custos de infraestrutura em mais de 93% em comparação a casos base não hierárquicos (planos), principalmente pela minimização do tráfego de rede inter-regional. Ademais, a seleção otimizada de região alcançou uma redução total de 40,6% no tempo de treinamento em relação ao caso base plano, superando a alocação geográfica intuitiva em uma margem adicional de 11,6%. Simulações calibradas expandiram a análise para 2.048 clientes, mostrando uma redução de 29,7% no tempo de comunicação intercontinental, confirmando a viabilidade e a eficiência da arquitetura proposta para cenários em escala global.

Palavras-chave: Aprendizado Federado Hierárquico, *Serverless*, Orquestração em Nuvem, Sistemas Geodistribuídos.

Abstract

The widespread adoption of Machine Learning (ML) models faces increasing scalability and privacy challenges, driven by the increased volume of generated data and strict data protection regulations. Federated Learning (FL) mitigates these risks by keeping data on local devices and transmitting only model weights. However, scaling FL to globally distributed clients introduces communication latency and transcontinental network bottlenecks. While Hierarchical Federated Learning (HFL) reduces this burden through intermediate edge aggregations, its deployment in traditional cloud computing environments typically relies on persistent (always-on) Infrastructure-as-a-Service (IaaS) virtual machines. This creates an operational bottleneck: the intermittent nature of FL training cycles leaves these provisioned resources idle for extended periods, resulting in high infrastructure costs and management overhead. To address these inefficiencies, this dissertation leverages the Serverless Computing paradigm to propose a Serverless HFL architecture. By abstracting underlying server management and executing tasks in ephemeral, event-triggered containers, this approach achieves a fully pay-per-use execution model that inherently scales to zero. Our architecture is orchestrated via managed cloud workflows, eliminating idle infrastructure costs while reducing the management burden of persistent orchestration. Furthermore, we introduce a region-aware optimization method to strategically place the global parameter server by balancing inter-continental network latency, data egress costs, and regional computation pricing. Empirical evaluation was conducted through real-world deployments on the Google Cloud Platform (GCP) using standard benchmarks (MNIST, CIFAR-10) and a world-scale geospatial case study (So2Sat) distributed across four continents. The results demonstrate that our hierarchical serverless approach reduces infrastructure costs by over 93% compared to non-hierarchical (flat) baselines, primarily by minimizing inter-regional network egress traffic. Moreover, the optimized region selection achieved a total reduction of 40.6% in training time relative to the flat baseline, outperforming intuitive geographic placement by an additional 11.6%. Calibrated simulations expanded the analysis to 2,048 clients, showing a 29.7% reduction in intercontinental communication time, confirming the viability and efficiency of the proposed architecture for global-scale scenarios.

Keywords: Hierarchical Federated Learning, Serverless, Cloud Orchestration, Geo-distributed Systems.

List of Figures

2.1	Centralized distributed training architecture. Raw data (represented by the colored file icons) is collected from multiple edge devices and transmitted to a central server for model training.	8
2.2	The classical Federated Learning (FL) aggregation process. Distributed clients transmit their local model weights ($w_1^t, w_2^t, \dots$) to the central parameter server, which performs a global aggregation to generate an improved model (W_g^{t+1}).	13
2.3	The three-tier Hierarchical Federated Learning (HFL) architecture. Edge servers act as regional hubs, consolidating local client updates (w_i^j) into intermediate regional models (W_e^{j+1}). The global server manages final convergence by periodically aggregating these regional models (W_e^t) into an updated global model (W_g^{t+1}).	15
2.4	Operational comparison between the standard Federated Averaging (FedAvg) and Hierarchical Federated Averaging (HierFAVG) algorithms. While FedAvg requires direct client-to-global synchronization, HierFAVG uses frequent regional aggregations (defined by κ_1 and κ_2) to reduce the frequency of global synchronizations.	18
2.5	The ontology of cloud computing proposed by Youseff et al. [68], organizing cloud services into layers. The upper layers correspond to the service models standardized by NIST, namely Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), while the lower ones add Data as a Service (DaaS), Communication as a Service (CaaS), and the underlying physical infrastructure.	21

4.1	Example of a flat Federated Learning (FL) process implemented as a server-less workflow. Yellow boxes represent stateless functions, while blue boxes denote control-flow steps managed by the workflow engine. The decision diamond evaluates the termination criterion, here the accuracy (Acc.) of the global model against a target value.	38
4.2	Proposed hierarchical workflow orchestration. The global workflow iterates over global rounds, invoking the edge workflows in parallel and aggregating the regional models they return. Each edge workflow, deployed in the region of the edge server it represents, iterates over the κ_2 regional rounds, selecting the participating clients, invoking the training function on each, and consolidating their updates into a regional model.	40
6.1	Geographical distribution of the 39 cities selected from the So2Sat dataset used in the Global Multi-Region experiment. In our architecture, clients representing these cities are mapped to and orchestrated by regional workflows in their nearest Google Cloud regions (Asia, Europe, South America, and United States).	51
6.2	Average wall-clock time comparison between the flat and the Hierarchical Federated Learning (HFL) configurations, for the MNIST and CIFAR-10 datasets, across different numbers of aggregation rounds.	53
6.3	Performance trade-offs to reach the target accuracy for MNIST and CIFAR-10, comparing the flat baseline against hierarchical configurations with different values of κ_1 and κ_2 . Bars represent the total wall-clock time (left axis), while the line tracks the number of global rounds required for convergence (right axis).	54
6.4	Infrastructure cost breakdown for CIFAR-10 reaching 70% accuracy. The comparison highlights the impact of hierarchical topology on network egress, aggregation function computation, and orchestration costs.	54
6.5	Performance trade-offs to reach the target accuracy for So2Sat, comparing the flat baseline against the intuitive and the optimized hierarchical placements of the global parameter server. Bars represent the total wall-clock time (left axis), while the line tracks the number of global rounds required for convergence (right axis).	56

-
- 6.6 Infrastructure cost breakdown for the So2Sat case study. The chart highlights the reduction in network egress costs achieved by the hierarchical architecture and the minimal overhead of serverless coordination. 57
- 6.7 Validation of the simulator against the empirical deployment on Google Cloud Platform (GCP), for the 39-client So2Sat case study. Each configuration, namely the flat baseline and the hierarchical (HFL) architecture under the intuitive (Int.) and the optimized (Opt.) placements of the global parameter server, is plotted twice in the plane of total execution time against infrastructure cost: as a circle for the value measured on the real infrastructure and as a cross for the value produced by the simulator. The dashed segment joining each pair therefore expresses the deviation between the two, and the shorter it is, the closer the simulator reproduces the observed behaviour. 60
- 6.8 Comparison of Hierarchical Federated Learning (HFL) strategies at scale, showing intercontinental communication time against total infrastructure cost, for 2,048 clients over 56 rounds of equivalent training effort. The always-on HFL configuration relies on persistent instances; Serverless HFL (SHFL) Int. replaces them with functions while retaining a persistent orchestrator and the intuitive global placement; and SHFL Workflows Opt. adds the workflow-based orchestration and the optimized global placement. 62

List of Tables

2.1	Inter-region data transfer (egress) rates on Google Cloud Platform (GCP) and Azure. GCP prices each source-destination pair individually, whereas Azure applies a single inter-continental rate determined by the source continent alone. Rates are per GiB for GCP and per GB for Azure.	25
2.2	Comparison of different choices for cloud as a service, adapted from Castro et al. [12].	28
3.1	Comparison of relevant works in Serverless Federated Learning and Hierarchical Federated Learning (HFL).	36
4.1	Equivalent managed services across GCP, AWS, and Azure, illustrating the cross-cloud portability of the proposed serverless stack.	43
A.1	Complete empirical results for the communication efficiency evaluation (corresponding to Figure 6.2), comparing the wall-clock time of the flat and the hierarchical architectures across different numbers of training rounds. .	73
A.2	Complete empirical results for the time-to-accuracy evaluation (corresponding to Figure 6.3), comparing the flat and the hierarchical architectures for MNIST (85% target) and CIFAR-10 (70% target).	76
A.3	Complete empirical results for the time-to-accuracy evaluation of the So2Sat global case study (corresponding to Figure 6.5), comparing the intuitive and the optimized placements of the global parameter server (50% target accuracy).	77
A.4	Measured inter-region communication latency (in milliseconds) used as the metric $\mathcal{L}$ of the region selection algorithm. Each value is the average of five executions of the metering function deployed in the source region, downloading a test payload from a storage bucket in the destination region. Rows denote the source region and columns the destination.	77

-
- A.5 Regional computation cost used as the metric $\mathcal{P}$ of the region selection algorithm, collected from the Google Cloud Run pricing tables. Values are expressed in US dollars per second of execution under the request-based billing model, the one adopted in this work. 78
- A.6 Inter-region egress rates used as the metric $\mathcal{E}$ of the region selection algorithm, in US dollars per GiB. Because Google Cloud Platform (GCP) prices each transfer by the pair of continents involved, the complete region-to-region matrix collapses to the values below. Rows denote the source continent and columns the destination. 78

Acronyms and Abbreviations

AWS	: Amazon Web Services
CaaS	: Communication as a Service
CPU	: Central Processing Unit
DaaS	: Data as a Service
EBS	: Elastic Block Store
EC2	: Elastic Compute Cloud
FaaS	: Function-as-a-Service
FedAvg	: Federated Averaging
FL	: Federated Learning
FTL	: Federated Transfer Learning
GCP	: Google Cloud Platform
GDPR	: General Data Protection Regulation
GPU	: Graphics Processing Unit
HaaS	: Hardware as a Service
HFL	: Hierarchical Federated Learning
HierFAVG	: Hierarchical Federated Averaging
HTTP	: Hypertext Transfer Protocol
IaaS	: Infrastructure-as-a-Service
IaC	: Infrastructure-as-Code
IDC	: International Data Corporation
IID	: Independent and Identically Distributed
IoT	: Internet of Things
LCZ	: Local Climate Zone
LGPD	: Lei Geral de Proteção de Dados
ML	: Machine Learning
NIST	: National Institute of Standards and Technology
PaaS	: Platform-as-a-Service
S3	: Simple Storage Service
SaaS	: Software-as-a-Service

SHFL	:	Serverless Hierarchical Federated Learning
SLA	:	Service-Level Agreement
SMC	:	Secure Multiparty Computation
TCP	:	Transmission Control Protocol
vCPU	:	Virtual Central Processing Unit
VM	:	Virtual Machine

Glossary

α, β, γ	: tunable weights for the region selection scoring function;
α_{Dir}	: concentration parameter of the Dirichlet distribution used to induce the non-IID partition;
BW	: nominal network bandwidth;
C_i	: distributed client node i ;
$\mathcal{C}_e$	: set of clients associated with edge parameter server e ;
d_i	: private local dataset held by client i ;
e	: index of an edge parameter server;
$\mathcal{E}^{(e)}(r)$	: egress rate, in US dollars per GiB, between candidate global region r and the region of edge server e ;
η	: network throughput penalty factor;
g	: cloud region hosting the global parameter server;
i	: index of a client;
k	: number of global communication rounds in a non-hierarchical (flat) architecture;
κ_1	: number of local training steps performed before an edge aggregation;
κ_2	: number of edge aggregation rounds performed before a global synchronization;
L	: total number of edge parameter servers;
$\mathcal{L}(a, b)$	: communication latency between region a and region b ;
$\mathcal{L}^{(e)}(r)$	: communication latency between candidate global region r and edge server e ;
$\mathcal{L}_{base}$	: intra-region baseline communication latency;
M_{FED}	: model collaboratively trained by the federation;
n	: total number of samples across the federation;
n_e	: total number of samples held by the clients of edge server e ;
n_i	: number of samples held by client i ;
N	: total number of distributed clients in the federation;

N_e	: number of clients connected to edge server e ;
$\text{Norm}(x)$	: min-max normalization of a metric value x ;
$\mathcal{P}(r)$	: computation cost within candidate cloud region r ;
r	: candidate cloud region for hosting the global parameter server;
$\mathcal{R}$	: set of all available cloud regions;
S_{mod}	: size of the machine learning model parameters;
t	: index of the current global communication round;
$t_{comm}(a, b)$	: network communication time between region a and region b ;
t_{proc}^{client}	: computational processing time for local client training;
t_{proc}^{edge}	: computational processing time for the edge aggregation function;
t_{proc}^{global}	: computational processing time for the global aggregation function;
$T_{edge}(e)$	: total time of the κ_2 regional rounds performed by edge server e ;
T_{round}^{flat}	: total execution time per round for the flat architecture;
T_{round}^{hier}	: total execution time per round for the hierarchical architecture;
θ	: parameter vector of a machine learning model;
W_e^t	: regional model weights of edge server e at global round t ;
W_g^t	: global model weights at global round t ;
w_i^t	: local model weights of client i at round t ;
$\mathcal{X}, \mathcal{Y}, \mathcal{I}$	: feature, label, and sample identifier spaces of a client's dataset;

Contents

1	Introduction	1
1.1	Dissertation Overview	4
2	Background	6
2.1	Federated Learning (FL)	6
2.1.1	Machine Learning (ML)	6
2.1.2	Definition	8
2.1.3	Entities	9
2.1.4	Data Partitioning	9
2.1.5	Scale of Federation	11
2.1.6	Challenges	12
2.1.7	Federated Averaging (FedAvg)	13
2.2	Hierarchical Federated Learning (HFL)	14
2.2.1	Definition	14
2.2.2	Entities	16
2.2.3	Hierarchical Federated Averaging (HierFAVG)	17
2.3	Cloud Computing	19
2.3.1	Definition	20
2.3.2	Service Models	20
2.3.3	Deployment Models	22
2.3.4	Cloud Regions and Egress Costs	23
2.4	Serverless Computing	24

2.4.1	Definition	25
2.4.2	Function-as-a-Service (FaaS)	26
2.4.3	Object Storage	28
2.4.4	Serverless Workflows	29
3	Related Work	32
3.1	Serverless FL	32
3.2	Hierarchical FL (HFL)	33
3.3	Serverless HFL (SHFL)	34
3.4	Globally Distributed FL	35
3.5	Summary and Comparative Analysis	35
4	Workflow-Based Orchestration	37
4.1	Serverless Orchestration Model	38
4.2	Hierarchical Coordination Strategy	39
4.3	Implementation	42
5	Global Server Placement	44
5.1	Evaluation Metrics	44
5.1.1	Network egress rate ($\mathcal{E}$)	45
5.1.2	Communication latency ($\mathcal{L}$)	45
5.1.3	Computation cost ($\mathcal{P}$)	46
5.2	Region Selection Score	46
6	Results	48
6.1	Experimental Validation	48
6.1.1	Experimental Setup	49
6.1.1.1	Datasets and Models	49

6.1.1.2	Data Partitioning	50
6.1.1.3	Regional Setup and Resource Allocation	51
6.1.2	Communication Efficiency	51
6.1.3	Time-to-Accuracy	52
6.1.4	Placement Optimization and Global Case Study	55
6.2	Simulated Validation	57
6.2.1	Wall-Clock Time Modeling	58
6.2.2	Infrastructure Cost Modeling	59
6.2.3	Simulator Validation	60
6.2.4	Large-Scale Experiment	61
7	Conclusion and Future Work	63
	Bibliography	66
	Appendix A – Complete Experimental Data	73
	Appendix B – Publications	79

Chapter 1

Introduction

Machine learning (ML) has become widely adopted across modern industry, leveraging the massive influx of user-generated data to create models that learn complex patterns [48]. Concurrently, according to the International Data Corporation (IDC), the global volume of generated data is expanding exponentially, driven primarily by the proliferation of edge and IoT devices [64]. Coping with this vast data generation challenges traditional centralized model training paradigms, exposing scalability limitations.

Beyond these scalability constraints, centralized architectures face privacy and policy hurdles [17]. Because they necessitate the transfer and storage of data at a central processing server, they conflict with the enactment of regulatory frameworks, such as the General Data Protection Regulation (GDPR) in Europe and the General Data Protection Law (*Lei Geral de Proteção de Dados*, LGPD) in Brazil [61]. These data protection laws have reshaped how organizations are legally permitted to collect and process user information, while controversies such as the UK DeepMind scandal (involving the mishandling of sensitive patient health records [62]) further illustrate the growing consensus that data privacy must be treated as a requirement in modern system design.

In scenarios requiring the collaborative training of a ML model with data distributed across multiple devices or organizations, such as hospitals collaborating to analyze sensitive medical records or smart retail platforms providing personalized product recommendations [67], traditional centralized approaches require heterogeneous edge devices to transmit their raw, locally generated data to a central processing server. This architecture exposes sensitive information and can overwhelm available network bandwidth [14]. To address these limitations, Federated Learning (FL) [48] has emerged as a decentralized alternative.

In FL, collaborative model training is achieved by retaining raw data on-device; clients perform local training and exchange only the updated model parameters (*weights*) with a central server for aggregation. In ML models, these weights are the numerical matrices that dictate how inputs are transformed into predictions, encapsulating the underlying patterns and knowledge extracted from the data. By transmitting only these learned parameters instead of raw data, FL effectively mitigates privacy risks while distributing the computational load across the edge. The central server then consolidates these distinct updates from the participating clients to form a new, improved global model.

In parallel, serverless computing has emerged as a paradigm shift in cloud application deployment. It is defined as a platform that abstracts server management from developers, running code on-demand with automatic scaling and billing tied to actual execution time and resource usage [12]. This model provides elasticity, allowing applications to scale seamlessly from zero to accommodate high-demand workloads while charging users exclusively for the what they actively use. Its primary embodiment, *Function-as-a-Service* (FaaS), operates by treating a function as the fundamental unit of computation, which is executed solely in response to triggers such as events or HTTP requests [12].

By abstracting away operational concerns such as availability and resource overprovisioning, this approach is positioned to address the intermittent communication cycles inherent to FL. In these systems, execution occurs in distinct bursts; the global server remains idle while distributed clients perform local training, and conversely, clients wait inactive while the server aggregates updates. Consequently, by executing orchestration and aggregation logic strictly upon invocation, this paradigm not only eliminates the idle-time infrastructure costs typical of traditional serverfull architectures but also scales to accommodate variable client participation across regions. However, while FaaS ensures billing only during active computation periods, it often carries a higher price per unit of time than continuously provisioned instances. This performance-cost tradeoff [35] motivates the design of more elastic and efficient FL systems.

Beyond optimizing the compute infrastructure, scaling FL systems to a global level exposes new bottlenecks. Geographically dispersed applications introduce spatial and network topology constraints. For instance, in mobile keyboard applications [29], such as training next-word prediction models for smartphones, the system must coordinate learning across millions of client devices. To protect user privacy and adhere to data minimization principles, these models must be trained locally on the devices without exporting sensitive user data to centralized servers [29]. Similarly, in broader industrial

applications, data often exists in isolated islands across different organizations that cannot directly share information due to privacy regulations. Establishing collaborative data networks among these international entities requires robust decentralized learning frameworks [67].

In these global scenarios, whether coordinating millions of smartphones or connecting international organizational silos, the central server that consolidates the updates, hereafter referred to as the *global parameter server*, tends to be distant from a substantial portion of the clients it serves, so the communication between them incurs transcontinental latency. To mitigate these communication bottlenecks, Hierarchical Federated Learning (HFL) introduces an intermediate tier of *edge parameter servers* that perform partial model aggregations closer to the clients [1, 44]. This approach leverages regional low-latency communication to reduce long-haul data transfers while maintaining access to global datasets, providing a foundation for more scalable and efficient distributed training.

To tackle these combined architectural constraints, recent work has integrated serverless computing with hierarchical topologies [33, 59]. However, while these studies provide significant advancements in FL algorithms, they do not address certain deployment and architectural strategies. Specifically, a naive selection of the cloud region to host the global parameter server can result in increased communication latency with edge servers. Furthermore, the common reliance on persistent orchestration entities introduces architectural dependencies that remain inconsistent with a fully serverless, pay-per-use execution model.

The primary contributions of this research can be articulated as follows:

- We conduct a comparative analysis between hierarchical and non-hierarchical (flat) topologies in a serverless environment and identify a trade-off between edge-level and global-level updates. Our findings show that while frequent edge rounds minimize expensive edge-to-global communication, they introduce cumulative orchestration and cold start overheads that must be balanced.
- We develop a method for selecting the optimized deployment region for the global parameter server relative to regional edge servers. This algorithm strategically selects the cloud hosting region by balancing inter-continental network latency, data egress costs, and regional computation pricing, outperforming intuitive region selection.
- We introduce a fully serverless coordination strategy for HFL orchestrated via man-

aged cloud workflows. This design replaces persistent orchestrators with a complete pay-per-use model, leveraging native cloud scalability to eliminate the management and financial burden of always-on infrastructure.

We validate our coordination strategy and optimization method through empirical cloud deployments conducted on a commercial public cloud provider, including a case study with the So2Sat dataset [69], which provides a realistic geographic distribution of clients. To the best of our knowledge, this is the first work to propose a fully serverless orchestration layer using workflow engines to coordinate FL processes. Complementarily, to evaluate the system’s scalability beyond the operational and financial constraints of wide-scale deployment, we developed a simulator calibrated with data collected from our actual cloud executions. Our evaluation demonstrates the performance and cost-effectiveness of combining a hierarchical topology with serverless orchestration and strategic region selection for geographically dispersed scenarios.

1.1 Dissertation Overview

The remainder of this dissertation is organized as follows. Chapter 2 establishes the technical foundations on which the work rests, drawing together the two domains it bridges. On the learning side, it presents FL and its hierarchical extension, along with the FedAvg and HierFAVG algorithms that govern them. On the infrastructure side, it reviews *cloud computing*, the organization of its infrastructure into geographic regions and the egress fees charged on the traffic that crosses between them, and the serverless paradigm, whose constituent services supply the execution model that the proposed architecture is assembled from.

Chapter 3 reviews the literature on serverless FL, hierarchical topologies, and the frameworks that combine the two, and identifies the gap this work addresses: those frameworks execute the training and the aggregation as functions, yet retain a persistent entity to orchestrate the rounds, and none of them treats the region hosting the global parameter server as a decision to be made.

Chapters 4 and 5 present the two contributions that follow from this gap. The former proposes an orchestration model in which the responsibilities of a persistent parameter server are decomposed into managed services, with the coordination of the federation expressed as nested, regionally deployed workflows. The latter proposes a region-aware placement method for the global parameter server, defining the three metrics against

which a candidate region is evaluated, namely inter-continental latency, network egress fees, and regional computation pricing, and the scoring function that weighs them against one another.

Chapter 6 evaluates both contributions, first through real deployments on GCP, culminating in a world-scale case study distributed across four continents, and then through a simulator calibrated with data collected from those executions, which extends the analysis to a federation of thousands of clients. Chapter 7 summarizes what the work establishes, delimits the conditions under which the proposed architecture pays off, and outlines the directions it leaves open. Appendix A reports the complete data underlying the evaluation, and Appendix B lists the publications produced over the course of this master's program.

Chapter 2

Background

This chapter establishes the technical foundations on which this work builds, drawing together the two domains that its contribution bridges: distributed machine learning and cloud infrastructure. We begin with FL, the paradigm that enables collaborative model training without centralizing private data, and then present its hierarchical extension, which introduces an intermediate aggregation tier to alleviate the communication bottlenecks that arise when the federation is dispersed across the globe. We then turn to the infrastructure side, reviewing the principles of cloud computing that underpin the deployment environment of this work, followed by the serverless computing paradigm, whose pay-per-use execution model motivates the architecture proposed in the following chapters.

2.1 Federated Learning (FL)

This section establishes the foundations of FL. We begin by reviewing the ML fundamentals underlying the paradigm, followed by its definition and the entities that compose a federated system. We then examine how such systems are characterized, in terms of data partitioning and the scale of the federation, and discuss their main challenges. Finally, we detail *Federated Averaging* (FedAvg), the algorithm that originated the paradigm and provides the basis for the hierarchical approach adopted in this work.

2.1.1 Machine Learning (ML)

ML provides the foundation upon which federated learning is built. A widely adopted definition is given by Mitchell [57]: “A computer program is said to learn from experience

E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ". In other words, rather than being explicitly programmed to solve a task, the program improves its behavior by extracting patterns from data.

This data is organized into *samples*, each corresponding to an individual observation. Every sample is described by a set of *features*, the input attributes that characterize it, and, in the supervised setting adopted in this work, by a *label*, the expected output the model should predict. In an image classification task, for instance, the features are the pixel values of the image and the label is the class it depicts. Each sample is further associated with an *identifier* that distinguishes it within the dataset, such as the customer to whom a bank record belongs.

As this work adopts neural networks, the foundational concepts below are presented following the deep learning literature [19]. The outcome of this learning process is a *model*: a mathematical function that maps the features of a sample to a prediction, such as assigning a label to an image. The model's behavior is encoded in its *parameters*, the internal values that are adjusted during training to fit the data. Concretely, a model defines a family of functions $f(x; \theta)$ mapping an input x to a prediction, where the parameter vector θ determines the specific mapping.

Training consists of searching for the parameter values that best fit the data. This search is often guided by a *loss function*, which quantifies the discrepancy between the model's predictions and the labels of the training samples; learning is thus framed as the problem of minimizing this loss over the training data. Different model families minimize this loss in different ways. Neural networks, the family adopted in this work, rely on *gradient descent*, which iteratively adjusts the parameters in the direction that most reduces the loss [19]. This process is governed by *hyperparameters*, such as the learning rate, which are set prior to training rather than learned from the data.

In a neural network, the parameters are commonly referred to as *weights* and are organized into matrices or tensors arranged across successive layers. These layers progressively transform the input into increasingly abstract representations until a final prediction is produced, and it is by adjusting their weights during training that the model learns to represent the underlying patterns of the data.

2.1.2 Definition

When the data required to train a model originates across multiple distributed devices, traditional ML training relies on a centralized architecture, as illustrated in Figure 2.1. In this paradigm, devices collect and transmit their raw, locally generated data to a central processing server, which aggregates these disparate datasets into a unified repository to perform model training.

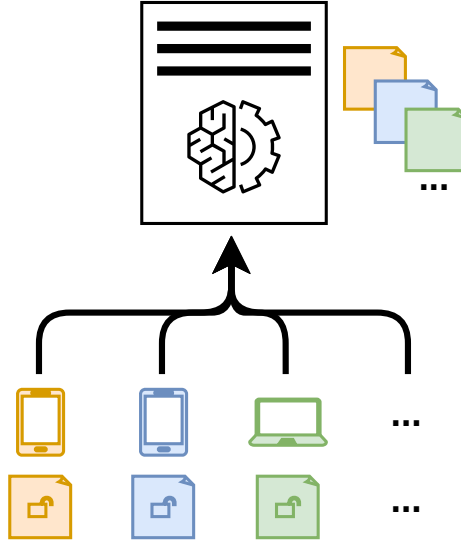


Figure 2.1: Centralized distributed training architecture. Raw data (represented by the colored file icons) is collected from multiple edge devices and transmitted to a central server for model training.

While straightforward, this approach faces scalability and privacy challenges. As the volume of data grows, the continuous transmission of raw information leads to network bottlenecks, excessive bandwidth consumption, and high latency [14]. Moreover, the necessity of centralizing sensitive user information creates privacy vulnerabilities and conflicts with strict regulatory frameworks that govern data processing and ownership, such as the GDPR and LGPD [17, 61].

FL [48] reverses this by shifting the training process from the server to the edge. Instead of moving raw data to the model, FL brings the model to the data. In this collaborative paradigm, edge devices (*clients*) retain their private data locally, performing training on-device and exchanging only the learned model parameters (weights w_i) with the central server.

Yang et al. [67] formalize this paradigm as follows. Consider a set of N clients $\{C_1, \dots, C_N\}$, where each client C_i holds a private local dataset d_i , such that the full collection of datasets is $\{d_1, \dots, d_N\}$. Their objective is to train a ML model that leverages all

of their combined data. While a conventional method is to centralize data $d = d_1 \cup \dots \cup d_N$ to train a model M_{SUM} , an FL system is a process in which these devices collaboratively train a model M_{FED} , achieving this by exchanging model weights w_i rather than exposing their private data d_i . The accuracy of M_{FED} , denoted as V_{FED} , should be close to the performance of M_{SUM} , V_{SUM} ; letting δ be a non-negative real number, if

$$|V_{FED} - V_{SUM}| < \delta, \quad (2.1)$$

the FL algorithm is said to have δ -accuracy loss.

2.1.3 Entities

A FL system is composed of two fundamental types of entity: the clients and the central parameter server, the latter also referred to in the literature as the *manager* [42]. The clients are the data owners that participate in the federation; each one retains its private dataset locally and performs the on-device training that produces the model updates. The central parameter server is responsible for aggregating the updates produced by the clients and maintaining the global model throughout training.

This work adopts a centralized organization, in which the clients interact through a single central parameter server rather than directly among themselves. This is the most common arrangement in the literature and the one assumed throughout this work. Decentralized alternatives, in which clients communicate directly with one another, for instance through peer-to-peer or blockchain-based schemes, have also been proposed to remove the dependence on a central coordinator [42]; such designs, however, fall outside the scope of this work.

The three-tier hierarchical organization adopted in this work, which introduces an intermediate aggregation layer between the clients and the central parameter server, builds directly upon these roles and is detailed in Section 3.2.

2.1.4 Data Partitioning

The federated paradigm can be realized under different configurations, depending on how data are distributed among the participating clients. Following Yang et al. [67], let the data held by each client be characterized by three spaces: the feature space $\mathcal{X}$, the label space $\mathcal{Y}$, and the sample identifier space $\mathcal{I}$. For any two clients C_i and C_j , the way in which their datasets d_i and d_j overlap across these spaces gives rise to three distinct

categories: horizontal, vertical, and federated transfer learning [67, 42].

In *horizontal* (or sample-based) FL, the clients share the same feature space but differ in their samples, that is,

$$\mathcal{X}_i = \mathcal{X}_j, \quad \mathcal{Y}_i = \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j, \quad \forall d_i, d_j, i \neq j. \quad (2.2)$$

An example is two regional banks that serve distinct user populations but record the same set of attributes for each customer. Because the clients describe their data through identical features, they can train local models with a common architecture and combine them directly through weight averaging.

In *vertical* (or feature-based) FL, the clients share the same sample space but differ in their features,

$$\mathcal{X}_i \neq \mathcal{X}_j, \quad \mathcal{Y}_i \neq \mathcal{Y}_j, \quad \mathcal{I}_i = \mathcal{I}_j, \quad \forall d_i, d_j, i \neq j. \quad (2.3)$$

For example, a bank and an e-commerce company operating in the same city may have largely overlapping customer sets, while each party holds different attributes about those shared users. Training in this setting requires aligning the common entities and jointly computing the model across the complementary feature sets, rather than averaging independent local models.

Finally, *Federated Transfer Learning* (FTL) applies when the clients differ in both the sample and feature spaces,

$$\mathcal{X}_i \neq \mathcal{X}_j, \quad \mathcal{Y}_i \neq \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j, \quad \forall d_i, d_j, i \neq j. \quad (2.4)$$

This corresponds, for instance, to institutions located in different regions and operating in different domains, where only a small portion of both users and features overlaps. In such cases, transfer learning techniques are employed to learn a shared representation across the parties.

Beyond how the feature and sample spaces are partitioned, a further distinction concerns the statistical distribution of the data across clients. In the ideal case, the data are assumed to be independent and identically distributed (IID), meaning that each client's local dataset can be regarded as a representative sample drawn from the same underlying distribution. In practice, however, this assumption rarely holds: because each client collects its own data under different conditions, local datasets are typically non-independent and identically distributed (non-IID), differing in aspects such as class proportions or fea-

ture distributions [42]. For instance, in an image classification task, one client may hold examples concentrated in a few classes while another observes an entirely different set, so that the local distributions diverge substantially from the global one.

This work operates within the horizontal setting, which is the most prevalent and extensively studied data partitioning in the FL literature [42]. Beyond its prevalence, this configuration is the one that best fits the geo-distributed scenarios targeted in this work, where clients spread across regions observe a shared feature space. Within this setting, both IID and non-IID data distributions are considered in the experimental evaluation.

2.1.5 Scale of Federation

Beyond how data are partitioned, a FL system is also characterized by the scale of the federation, that is, the number of participating clients and the nature of each one. The literature identifies two regimes, *cross-silo* and *cross-device*, which differ in the number of clients and the amount of data and computational power held by each [42].

In the *cross-silo* setting, the clients are organizations or data centers. The federation comprises a relatively small number of participants, each holding a large volume of data and substantial computational resources. These clients are stable and reliable, and can be expected to participate continuously throughout the training process. This regime underlies collaborative scenarios across institutional boundaries: hospitals jointly training a diagnostic model over their private medical records without sharing patient data, or banks and e-commerce platforms cooperating to improve recommendation or risk-assessment models, such as detecting multi-party borrowing across institutional boundaries [67, 42].

In the *cross-device* setting, conversely, the federation involves a very large number of clients, often reaching millions, each holding a small amount of data and limited computational power. The clients are typically mobile or edge devices, characterized by unstable connectivity and intermittent availability. Because it is impractical to assume that all devices participate in every round, the system selects only a fraction of clients at each iteration and must tolerate frequent disconnections [42]. A representative application is next-word prediction for mobile keyboards, where millions of smartphones collaboratively improve a shared language model while keeping users' typed data on-device [29].

The scenarios targeted in this work span both regimes: the geo-distributed federation may consist of a moderate number of large regional participants or a vast population of edge clients, with the latter being the more demanding case in terms of scale and

communication.

2.1.6 Challenges

Despite its advantages, FL introduces operational and security challenges. It is important to acknowledge that the exchange of model weights does not guarantee absolute privacy. Although FL avoids the direct sharing of raw data, research indicates that intermediate model parameters can inadvertently reveal sensitive information [67]. This passive leakage manifests through inference attacks, in which malicious actors reconstruct features or determine whether specific samples were included in a client’s training set.

Beyond passive leakage, the system remains vulnerable to active adversarial actions that can be mounted at distinct stages of the training process [42]. At the input stage, compromised clients may perform *data poisoning* by manipulating their local training samples; during the learning process, they may conduct *model poisoning* by uploading maliciously crafted updates, including the insertion of “backdoors” into the global model, or behave arbitrarily as Byzantine participants. Consequently, while FL significantly limits the attack surface compared to centralized paradigms, ensuring privacy and security requires the integration of additional mechanisms, such as Secure Multiparty Computation (SMC), Differential Privacy, or Homomorphic Encryption, to prevent indirect information leakage [67, 42].

Aside from privacy concerns, the deployment of an federated learning system requires addressing system-level bottlenecks regarding efficiency, effectiveness, and autonomy [42]. Efficiency is constrained by the communication overhead inherent to multi-round parameter synchronization across geographically dispersed nodes. Effectiveness is often challenged by the statistical heterogeneity of local data (non-IID distributions), which can degrade model convergence and final accuracy. Finally, ensuring system autonomy remains a significant hurdle, as practical architectures must handle participant unreliability, network disconnections, and dynamic scheduling without derailing the overall training lifecycle. This last concern is not exclusive to FL, but common to distributed learning at the edge: in their systematic review of the field, Poncinelli Filho et al. [16] identify the handling of device failures among its core open challenges, and observe that it remains among the least addressed by the literature they surveyed.

2.1.7 Federated Averaging (FedAvg)

The training process in FL is classically governed by the FedAvg algorithm [48], the method that originated the paradigm. As depicted in Figure 2.2, the central parameter server orchestrates an iterative training cycle. In each communication round, denoted by the index t , the server broadcasts the current global model weights W_g^t to a subset of clients. These clients perform local training steps on their private data to compute updated local weights $\{w_1^t, w_2^t, \dots, w_N^t\}$, which are then transmitted back to the central parameter server.

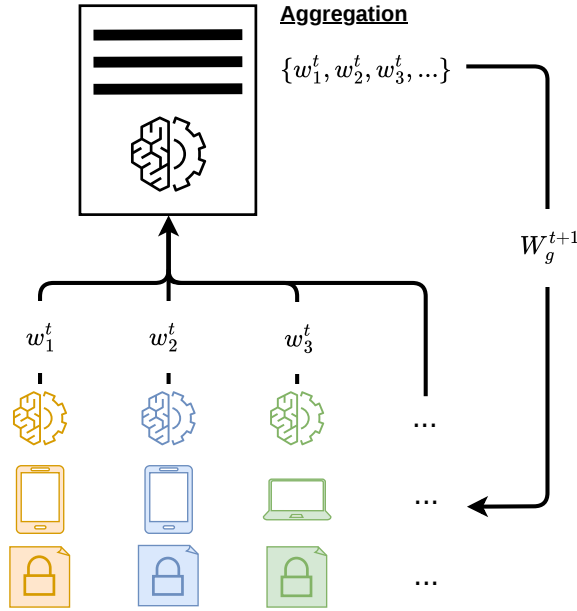


Figure 2.2: The classical Federated Learning (FL) aggregation process. Distributed clients transmit their local model weights ($w_1^t, w_2^t, \dots$) to the central parameter server, which performs a global aggregation to generate an improved model (W_g^{t+1}).

As its name suggests, FedAvg consolidates these updates through a weighted average, where each client's contribution is weighted by the size of its local dataset. Let $n_i = |d_i|$ denote the number of samples held by client C_i and $n = \sum_{i=1}^N n_i$ the total number of samples across the federation. The improved global model for the subsequent round is then computed as

$$W_g^{t+1} = \sum_{i=1}^N \frac{n_i}{n} w_i^t. \quad (2.5)$$

By weighting each local model proportionally to its dataset size, clients holding more data exert a greater influence on the aggregated model. This averaging-based aggregation ensures that the model converges without the need for raw data exchange, mitigating

privacy risks while distributing the computational load across the edge. This two-tier topology, in which every client synchronizes directly with a single central parameter server, is referred to throughout this work as the *flat* setting.

Although FedAvg remains the canonical algorithm of FL, its limitations under realistic conditions have motivated a broad family of extensions, surveyed by Li et al. [42]. Some target the statistical heterogeneity of client data, refining the local training or the aggregation step to improve convergence under non-IID distributions; others focus on reducing the communication burden of the synchronization rounds, for instance through update compression and quantization. Among these directions, a particularly relevant one for geo-distributed settings replaces the flat client-to-server topology with a hierarchical structure, introducing intermediate aggregation tiers that localize communication [44]. This hierarchical formulation, which forms the basis of the architecture proposed in this work, is detailed in the following section.

2.2 Hierarchical Federated Learning (HFL)

Building on the foundations of FL, this section presents the hierarchical extension that underpins the architecture proposed in this work. We begin by defining HFL and the three-tier organization that distinguishes it from the flat setting. We then describe the entities that compose a hierarchical federation and the role each one plays. Finally, we detail HierFAVG, the algorithm that governs the training process.

2.2.1 Definition

The FedAvg algorithm introduced in Section 2.1.7 assumes a flat, two-tier topology in which every client synchronizes directly with a single central parameter server. As observed by Liu et al. [44], this server can be positioned at two extremes, each with a structural drawback. When it resides in the cloud, it reaches a large population of clients and thus accesses the volume of data needed to train deep models; the communication with the cloud, however, is slow and often unpredictable, for instance due to network congestion, which renders each synchronization round costly. When it resides at the network edge, in contrast, the server communicates more efficiently with its nearby clients, but the limited number of devices within its reach restricts the accessible data and degrades the quality of the resulting model.

HFL reconciles these two extremes by inserting an intermediate aggregation tier be-

tween the clients and the global parameter server [44]. It organizes the federation as a three-tier architecture composed of clients, edge parameter servers, and a single global parameter server. Rather than trading broad data access for cheap communication, this arrangement localizes the most frequent synchronization to high-speed regional links while preserving access to the full data distribution through periodic global consolidation. This organization is illustrated in Figure 2.3. Although the top tier is traditionally referred to as a *cloud parameter server* in the literature [44], we adopt the term global parameter server throughout this work to avoid ambiguity, as the entire system is deployed within a public commercial cloud infrastructure. The interposition of such a tier is not peculiar to HFL: as Poncinelli Filho et al. [16] document in their review of distributed learning at the edge, several federated architectures place an intermediate layer between the end devices and the cloud, whether to absorb the coordination that neither extreme handles well or to distribute the aggregation across distinct hierarchies of nodes.

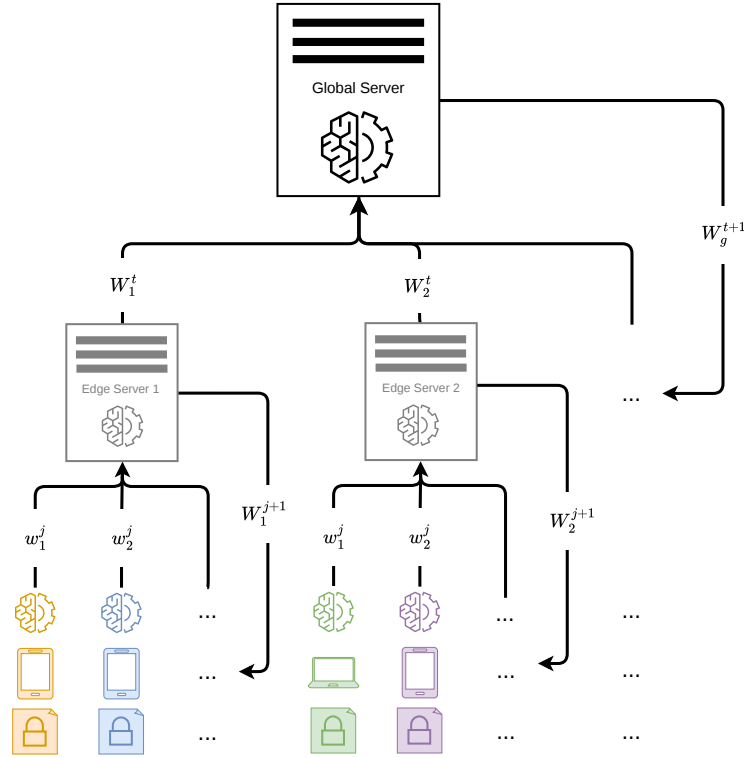


Figure 2.3: The three-tier Hierarchical Federated Learning (HFL) architecture. Edge servers act as regional hubs, consolidating local client updates (w_i^j) into intermediate regional models (W_e^{j+1}). The global server manages final convergence by periodically aggregating these regional models (W_e^t) into an updated global model (W_g^{t+1}).

Beyond its operational appeal, this intermediate tier rests on established theoretical guarantees. The hierarchical formulation preserves the convergence of FedAvg for both

convex and non-convex objectives, the latter covering the neural networks adopted in this work, even under non-IID data across clients [44]. The regional aggregation step does more than curb communication. By averaging models locally before each global round, it attenuates the impact of statistical heterogeneity across clients, mitigating the convergence degradation typically caused by non-IID data [65]. Moreover, for a fixed number of global rounds, shifting aggregation toward the edge can reduce the number of local iterations required to reach a target accuracy [44].

HFL extends the FL paradigm defined in Section 2.1.2 by introducing a regional partition of the federation. The N clients $\{C_1, \dots, C_N\}$ are grouped among L edge servers, such that each edge server e is associated with a disjoint subset of N_e clients $\{C_1^e, \dots, C_{N_e}^e\}$, and every client belongs to exactly one region. Each edge server thus holds a regional dataset $\{d_1^e, \dots, d_{N_e}^e\}$, formed by the private data of its member clients. The collaborative objective remains that of the flat setting, jointly training a model M_{FED} without exposing the private data d_i , but the interaction between the clients and the global parameter server is now mediated by this intermediate regional layer, rather than established directly. It is this partition, rather than any change to the learning goal, that distinguishes HFL from the flat setting.

2.2.2 Entities

The hierarchical paradigm builds upon the two entities of the flat by inserting a third one between them, resulting in three types of participant: clients, edge parameter servers, and a single global parameter server. The clients retain the role described in Section 2.1.3: they are the data owners that train on their private data and produce the local updates, and remain unchanged by the hierarchy except that they now report to a nearby edge server rather than to the central one.

The edge parameter servers are the intermediate entities that the hierarchy introduces. Each one acts as a regional aggregation hub, consolidating the updates of the cluster of clients assigned to its region into a single regional model. Because these servers are placed in proximity to their clients, the frequent exchange of updates between the two occurs over high-speed, low-latency regional links, which is what allows the hierarchy to relieve the pressure on the long-haul connection to the global server. From the standpoint of its own clients, an edge server plays a role analogous to that of the central parameter server in the flat setting; the distinction is that it coordinates only a regional subset of the federation rather than its entirety.

The global parameter server sits at the top tier and preserves the role of the flat central server, now exercised over the edge servers instead of directly over the clients. It consolidates the regional models produced by the edge servers into a single global model and redistributes it across the federation, thereby maintaining access to the full data distribution while communicating only with the edge servers rather than with every client. This tier is the one whose placement this work optimizes, as its position relative to the edge servers governs the cost and latency of the infrequent, but expensive, long-haul synchronization.

2.2.3 Hierarchical Federated Averaging (HierFAVG)

Just as FedAvg governs training in the flat setting, the hierarchical organization is trained through HierFAVG [44], an extension of FedAvg to the three-tier architecture. The difference lies in the synchronization pattern: whereas FedAvg interleaves local client updates with a single aggregation step, HierFAVG introduces two nested aggregation cycles, so that the global model is updated only after a sequence of regional aggregation rounds. This scheme is what allows the frequent, low-cost regional synchronizations to absorb most of the coordination, relegating the expensive long-haul global synchronization to an infrequent step.

Two hyperparameters control the balance between local computation and communication frequency:

- κ_1 : the number of local updates each client performs before its weights are sent to the associated edge server for a regional aggregation;
- κ_2 : the number of regional aggregation rounds each edge server performs before its partially aggregated model is sent to the global server for a global aggregation.

Figure 2.4 contrasts the two protocols. Under FedAvg, every client performs a block of local updates and then synchronizes directly with the global server. Under HierFAVG, clients instead synchronize with their edge server every κ_1 local updates, and the global server is engaged only once every κ_2 regional rounds, that is, once every $\kappa_1 \times \kappa_2$ local updates. In the depicted example, both protocols perform six local updates between consecutive global aggregations, but HierFAVG breaks this interval into $\kappa_2 = 2$ regional rounds of $\kappa_1 = 3$ local updates each, interposing two low-cost edge aggregations where FedAvg performs none.

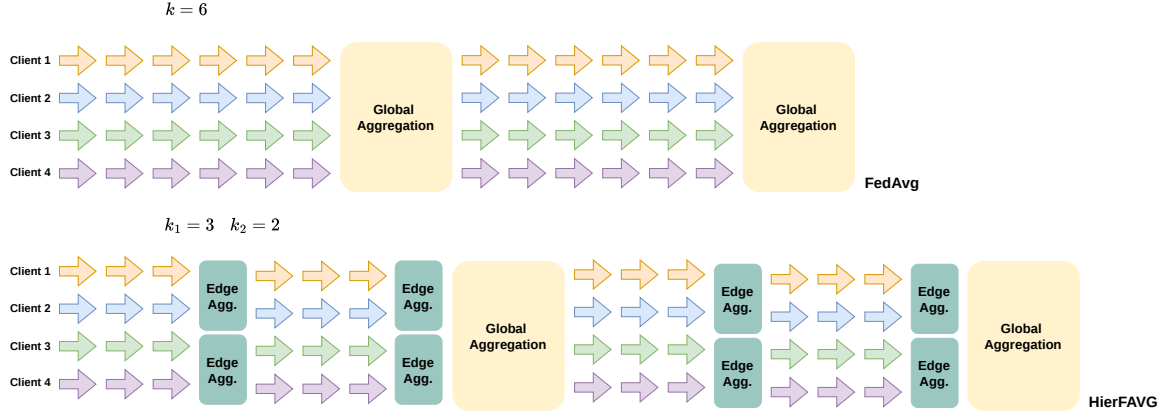


Figure 2.4: Operational comparison between the standard Federated Averaging (FedAvg) and Hierarchical Federated Averaging (HierFAVG) algorithms. While FedAvg requires direct client-to-global synchronization, HierFAVG uses frequent regional aggregations (defined by κ_1 and κ_2) to reduce the frequency of global synchronizations.

The procedure relies on two aggregation functions, one per tier, both computing a weighted average in which each contribution is weighted by the size of its underlying dataset, mirroring the FedAvg rule of Eq. 2.5. At the edge level, each server e aggregates the updates of its N_e clients into a regional model:

$$W_e = \sum_{i=1}^{N_e} \frac{n_i}{n_e} w_i, \quad n_e = \sum_{i=1}^{N_e} n_i, \quad (2.6)$$

where $n_i = |d_i|$ is the number of samples held by client i and n_e is the total number of samples in region e . At the global level, the parameter server consolidates the L regional models into a new global model, weighting each by the size of its aggregated dataset:

$$W_g = \sum_{e=1}^L \frac{n_e}{n} W_e, \quad n = \sum_{e=1}^L n_e, \quad (2.7)$$

with n the total number of samples across the federation. Setting $\kappa_2 = 1$ recovers standard FedAvg, as every regional round would trigger a global aggregation, collapsing the two tiers into one.

The full procedure is presented in Algorithm 1. Within each global round, the edge servers operate in parallel: each coordinates κ_2 regional rounds, and within every regional round its clients perform κ_1 local updates before an edge aggregation. Once the κ_2 regional rounds are complete, each edge server forwards its model to the global server, which produces the updated global model and broadcasts it back across the federation.

Algorithm 1 HierFAVG Procedure

```

1: for each global round  $t = 1, 2, \dots, T$  do
2:   for each edge server  $e \in \{1, \dots, L\}$  in parallel do
3:     for each regional aggregation round  $j = 1, \dots, \kappa_2$  do
4:       Clients  $i \in \{1, \dots, N_e\}$  perform  $\kappa_1$  local updates
5:        $W_e^j \leftarrow \text{EdgeAggregation}(\{w_1^j, \dots, w_{N_e}^j\})$ 
6:     end for
7:     Send  $W_e^t$  to global server
8:   end for
9:    $W_g^{t+1} \leftarrow \text{GlobalAggregation}(\{W_1^t, \dots, W_L^t\})$ 
10:  Broadcast  $W_g^{t+1}$  to all edge servers and clients
11: end for

```

Although Algorithm 1 is stated for a fixed number of global rounds T , training may equally be terminated by other criteria, such as reaching a target accuracy or exhausting a predefined time budget. In either case, the defining property of the hierarchy is that each global round involves an exchange between the global server and the L edge servers, rather than between the global server and all N clients. Since a federation typically comprises far more clients than regions ($L \ll N$), the volume of data crossing the long-haul links in a single global round is reduced accordingly. This structural saving does not come at the expense of learning: as established in Section 2.2.1, the regional aggregation that replaces the direct client-to-global exchange is what attenuates the effect of statistical heterogeneity, so that the reduction in long-haul traffic is obtained while preserving, and under non-IID data even improving, the convergence behavior of the flat protocol.

2.3 Cloud Computing

This section establishes the conceptual and structural foundations of cloud computing. We begin by defining the paradigm through academic and industry perspectives, followed by an analysis of the primary service models that classify cloud offerings based on their level of abstraction. We then examine the deployment models that dictate how cloud infrastructure is provisioned and managed. Finally, we describe how this infrastructure is organized into geographical regions and the cost implications of transferring data between them.

2.3.1 Definition

Cloud computing can be defined from various perspectives, ranging from its service delivery model to its underlying infrastructure. Armbrust et al. [9] define cloud computing broadly, encompassing both the applications delivered as services over the Internet and the hardware and systems software within the data centers that provide those services. From a systems architecture standpoint, Buyya et al. [11] describe a cloud as a parallel and distributed system consisting of interconnected and virtualized computers that are dynamically provisioned and presented as unified computing resources based on negotiated service-level agreements (SLAs).

Complementing these views, the National Institute of Standards and Technology (NIST) provides a widely accepted definition, characterizing cloud computing as a model that enables convenient, and on-demand network access to a shared pool of configurable computing resources (such as networks, servers, storage, applications, and services) [49]. A key aspect of the NIST definition is that these resources can be rapidly provisioned and released with minimal management effort or service provider interaction.

By synthesizing these definitions, a common denominator emerges: cloud computing represents an evolution in infrastructure where virtualized, distributed resources (both hardware and software) are pooled and delivered as on-demand, elastic services over the Internet. This paradigm shifts the burden of resource management and capacity planning away from the consumer, relying on automated provisioning and SLAs to deliver scalable computing.

2.3.2 Service Models

The architecture of cloud computing can be classified based on the level of abstraction provided to the consumer. Youseff et al. [68] proposed an ontology that classifies the cloud into a hierarchical structure, where the upper layers map directly to the three foundational service models standardized by the National Institute of Standards and Technology (NIST) [49]. This structure is illustrated in Figure 2.5.

According to the NIST definition, complemented by the layers defined in the ontology, cloud services are categorized into three primary models:

- **Software as a Service (SaaS):** Representing the topmost *Cloud Application* layer in the ontology, this model provides consumers with the capability to use the provider's

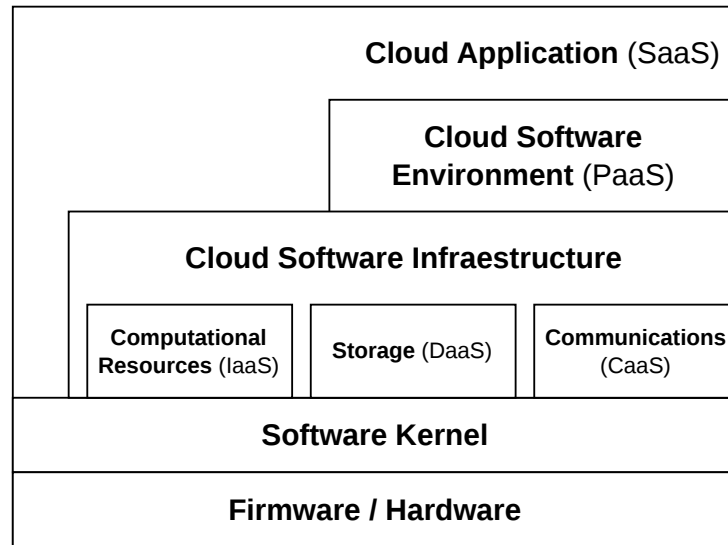


Figure 2.5: The ontology of cloud computing proposed by Youseff et al. [68], organizing cloud services into layers. The upper layers correspond to the service models standardized by NIST, namely Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), while the lower ones add Data as a Service (DaaS), Communication as a Service (CaaS), and the underlying physical infrastructure.

applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface (e.g., a web browser) or a program interface. Crucially, the consumer does not manage or control the underlying cloud infrastructure (including networks, servers, operating systems, storage, or even individual application capabilities), with the possible exception of limited user-specific application configuration settings.

- **Platform as a Service (PaaS):** Corresponding to the *Cloud Software Environment* layer, PaaS provides the capability to deploy onto the cloud infrastructure consumer-created or acquired applications using programming languages, libraries, services, and tools supported by the provider. In this model, the consumer does not manage or control the underlying cloud infrastructure, but maintains control over the deployed applications and possibly configuration settings for the application-hosting environment.
- **Infrastructure as a Service (IaaS):** Situated at the *Cloud Software Infrastructure* layer, this model abstracts the physical hardware and provides the consumer with provisioned processing, storage, networks, and other fundamental computing resources. The consumer does not manage or control the underlying physical cloud

infrastructure, but maintains control over operating systems, storage, deployed applications, and limited control of select networking components (e.g., host firewalls).

Beyond the three foundational models established by NIST, Youseff et al. [68] identifies additional service models that operate within the cloud architecture. Specifically, within the *Cloud Software Infrastructure* layer, the authors define Data as a Service (DaaS) and Communication as a Service (CaaS), which provide on-demand data storage and network routing capabilities, respectively. Furthermore, at the foundational physical level, the ontology outlines Hardware as a Service (HaaS), representing the leasing of raw hardware and data center facilities directly to consumers or other cloud operators.

2.3.3 Deployment Models

While service models define the level of abstraction provided to the consumer, deployment models dictate how the underlying cloud infrastructure is provisioned, managed, and accessed regarding exclusivity and physical location. According to the NIST definition [49], cloud infrastructure can be deployed across four primary models:

- **Private Cloud:** The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may exist on or off premises, and can be owned, managed, and operated by the organization, a third party, or some combination of them.
- **Community Cloud:** The infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations).
- **Public Cloud:** The cloud infrastructure is provisioned for open use by the general public. It exists on the premises of the cloud provider and may be owned, managed, and operated by a business, academic, or government organization.
- **Hybrid Cloud:** This model is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology enabling data and application portability.

Among these configurations, the *Public Cloud* has been central to the consolidation of the cloud computing paradigm, enabling the vision of computing as a broadly accessible

utility. Commercial public clouds, such as Amazon Web Services (AWS) [6], Google Cloud Platform (GCP) [26], and Microsoft Azure [56], implement the foundational service models (IaaS, PaaS, and SaaS) within their infrastructure.

At the IaaS layer, these providers offer on-demand, computing resources that allow consumers to deploy and run arbitrary software, including operating systems and applications. Specifically, computational instances at the IaaS tier are exemplified by AWS Elastic Compute Cloud (EC2) [3], while raw storage capabilities are provided by services such as Amazon Elastic Block Store (EBS) [2]. Moving up the abstraction stack to the platform level (PaaS), providers supply managed environments tailored for deploying web applications and services, such as Google App Engine [22]. These environments shield developers from the operational complexities of hardware and virtual machine management, while still requiring consumers to maintain control over application configuration settings and the hosting environment itself [49, 68].

Finally, at the highest application tier (SaaS), vendors such as Salesforce.com¹, alongside comprehensive productivity platforms like Google Workspace² and Microsoft 365³, among many other examples, deliver complete software applications directly to end-users via web portals, fully hosted and scaled by the underlying cloud infrastructure.

Crucially, across these service tiers, commercial public clouds are driven by a utility-based pricing model commonly referred to as “pay-as-you-go”. Under this model, consumers are billed based on the allocation time of resources (such as virtual machine uptime, measured in compute hours, or reserved storage volume), mirroring the delivery of traditional utilities like water or electricity [11]. However, a discrepancy exists: while a water utility charges for the exact volume consumed, traditional cloud billing is tied to resource provisioning rather than active processing. Consequently, consumers incur costs even during periods of application idleness, much like paying for a water tap to remain open regardless of whether any water is flowing. This limitation, where capacity is paid for but not utilized, motivates the more granular billing models examined in Section 2.4.

2.3.4 Cloud Regions and Egress Costs

The infrastructure of a commercial public cloud is not a single, monolithic data center but a collection of *regions*: independent, geographically dispersed sites, each corresponding

¹<https://www.salesforce.com>

²<https://workspace.google.com>

³<https://www.microsoft.com/microsoft-365>

to a set of facilities operated by the provider in a given geographic location [27, 55]. Providers such as AWS, GCP, and Azure operate dozens of such regions across multiple continents, allowing consumers to deploy their services close to end users or data sources. This geographic flexibility is what makes globally distributed deployments feasible, but it also turns the placement of a service across regions into a design decision with direct performance and cost implications.

One point among these implications is the cost of moving data between regions. Cloud providers charge *egress* fees for network traffic that leaves a region, and two properties of this pricing model are particularly consequential. First, egress is billed by the volume of data transferred rather than by the rate at which it is sent: transferring a given payload slowly or quickly incurs the same charge [34]. Second, the pricing is asymmetric: traffic entering a region (ingress) is typically free, while traffic leaving it is charged, so the cost of a data exchange is attributed entirely to its egress side.

Beyond these properties, the egress rate itself is not uniform: it depends on the geographic route the data follows. For transfers within a single provider’s network, exchanges between distant regions are priced higher than exchanges between nearby ones, reflecting the cost of maintaining the long-haul, transcontinental backbone that carries the traffic [34]. Table 2.1 illustrates this variability. On GCP, transferring data out of a North American region costs \$0.05 per GiB when the destination is in Europe, but rises to \$0.14 per GiB when the destination is in South America, nearly three times as much for the same volume of data, while transfers that remain within a single region incur no charge at all [27]. The table also shows that providers structure this pricing differently: whereas GCP prices each source-destination pair individually, Azure applies a single inter-continental rate determined solely by the source continent [55]. As a result, the placement of communicating services relative to one another governs not only the latency they experience but also the volume-based fees they accrue.

2.4 Serverless Computing

This section defines the serverless paradigm and the managed services through which it is realized. We begin with its defining characteristics and its position among the cloud service models, then examine FaaS, the service that supplies computation under this model. Because a function is stateless and executes in isolation, an application built on FaaS alone retains neither the data it operates on nor the logic that sequences its

Table 2.1: Inter-region data transfer (egress) rates on Google Cloud Platform (GCP) and Azure. GCP prices each source-destination pair individually, whereas Azure applies a single inter-continental rate determined by the source continent alone. Rates are per GiB for GCP and per GB for Azure.

Route	GCP [27]	Azure [55]
Within North America	\$0.02	\$0.02
Within Europe	\$0.02	\$0.02
Within South America	\$0.14	\$0.16
North America to Europe	\$0.05	\$0.05
North America to Asia	\$0.08	\$0.05
North America to South America	\$0.14	\$0.05
South America to North America	\$0.14	\$0.16

steps, and each of these must be restored by a further service of the same kind. The two subsections that follow describe them: object storage, which holds the state that functions cannot, and serverless workflow engines, which orchestrate the invocations that functions cannot coordinate among themselves.

2.4.1 Definition

The term *serverless computing* does not imply the absence of physical servers, but rather signifies an architectural shift concerning how these resources are managed. It introduces a “less thinking about servers” approach, where developers are relieved from low-level operational concerns, including server provisioning, monitoring, maintenance, and fault tolerance. Consequently, serverless computing is defined as an application deployment paradigm that entirely hides server management and usage from tenants, executing workloads dynamically on-demand [66, 12]. While early literature often equates serverless with FaaS, the paradigm extends beyond event-driven function execution to encompass other managed components, such as serverless databases, provided they adhere to the same operational abstractions.

This paradigm is defined by two core characteristics: transparent elasticity and a strict execution-based financial model [12]. In traditional cloud deployments, the “pay-as-you-go” model bills customers based on the allocation time of virtual resources, often leading to payments for idle capacity. Serverless computing refines this approach by billing users in fine-grained time units based solely on resources consumed during active execution.

This distinction is encapsulated by Paul Johnston, co-founder of ServerlessDays⁴, who defines a serverless solution as “one that costs you nothing to run if nobody is using it (excluding persistent data storage)” [38]. Consequently, this model supports “scaling to zero”, ensuring that applications are economically dormant during periods of inactivity. Operationally, the cloud provider assumes full responsibility for this elasticity, automatically scaling concurrent executions from zero to virtually infinity in response to workload fluctuations, without requiring developers to write auto-scaling policies.

The adoption of serverless can provide benefits for both cloud providers and developers [12]. From the developer’s perspective, it lowers the effort required to author, deploy, and manage cloud-scale applications by abstracting away the underlying infrastructure. Beyond operational simplicity, it introduces a financial advantage: because billing is tied to active execution, developers are freed from costs associated with overprovisioning and maintaining idle resources during periods of low or zero demand. Conversely, from the perspective of cloud providers, serverless computing presents an opportunity to better control the entire stack. By managing the execution environment at a granular level, providers can consolidate workloads, optimize scheduling, and reduce their own operational costs through more efficient utilization of their physical resources.

2.4.2 Function-as-a-Service (FaaS)

The most natural and widely adopted embodiment of serverless computing is FaaS. It is a cloud computing platform where the fundamental unit of computation is a short-running, stateless function executed in response to triggers [12]. Because tenants receive minimal access to the application’s underlying runtime configuration, FaaS allows developers to focus on authoring small, event-driven applications dedicated to specific tasks [66].

At its core, FaaS operates by executing code within isolated sandboxes or container technologies, such as Docker [50], to ensure tenant isolation in multitenant environments. These isolated execution environments are commonly referred to as *function instances* [66]. In a typical FaaS programming model, a function takes a dictionary as input and produces a dictionary as output. While some platforms simplify configuration by allowing tenants to define only the allocated main memory, automatically provisioning CPU power proportionally, this is not a universal constraint. As cloud platforms evolve, resource allocation has become increasingly granular; in providers like GCP, for instance, developers can explicitly and independently define memory, CPU allocations, and even

⁴<https://www.serverlessdays.io/>

provision hardware accelerators such as GPUs to meet specific workload demands [20]. Furthermore, the abstraction provided by FaaS hides a heterogeneous underlying physical infrastructure. As demonstrated in the empirical study by Wang et al. [66], providers schedule function instances across a diverse array of underlying virtual machine (VM) models.

The execution lifecycle of a FaaS application is governed by an event processing system [12]. Functions can be invoked either synchronously, where the caller waits for a response, or asynchronously, where execution is triggered as a background task. Upon the arrival of an event, the platform validates the request, checks resource limits, and queues the event. A worker node then allocates a function instance, injects the user’s code, and executes the function. This process of instantiating an entirely new function instance and setting up the runtime environment is an operation known as a *cold start*, which affects application responsiveness [66, 12]. To mitigate this latency, platforms attempt to perform *warm starts* by reusing existing idle function instances that have already been instantiated. After a period of inactivity, the platform will terminate the idle function instances to reclaim resources, erasing any temporary data stored on their non-persistent local disks. Consequently, FaaS functions are stateless; developers cannot rely on local memory or disk persistence across executions and must instead utilize external storage services to retrieve and update any required application state.

When evaluating these operational mechanics, FaaS platforms represent an architectural evolution of the PaaS service model [12]. In first-generation PaaS, the underlying hardware is abstracted, but developers still maintain control over the application-hosting environment configurations and must manually manage scaling rules. While traditional PaaS instances remain active and incur costs even when idle, FaaS pushes this abstraction further by removing user control over the hosting environment to provide transparent auto-scaling and a strictly usage-based billing model. This architectural shift is summarized in Table 2.2, which compares the level of abstraction, scaling mechanics, and cost structures across different cloud service models.

While elasticity and the ability to scale to zero are beneficial, the FaaS model introduces architectural and financial trade-offs. As previously noted, the reliance on external storage services to manage application state introduces additional network latency and shifts the burden of reliability and quality-of-service guarantees to these external systems. Financially, FaaS is billed in fine-grained increments (e.g., hundreds of milliseconds), meaning tenants do not pay for unused or idle resources [66, 12]. However, the raw unit

Table 2.2: Comparison of different choices for cloud as a service, adapted from Castro et al. [12].

	IaaS	1st Gen PaaS	FaaS (Serverless)
Expertise required	High	Medium	Low
Developer Control	High	Medium	Low
Scaling/Cost	Requires high-level of expertise to build auto-scaling rules and tune them	Requires high-level of expertise to build auto-scaling rules and tune them	Auto-scaling to work-load requested, and only paying for when running (scale to zero)
Unit of work	Low-level infrastructure building blocks (VMs, network, storage)	Packaged code that is deployed and running as a service	One function execution
Granularity of billing	Medium to large granularity: minutes to hours per resource	Medium to large granularity: minutes to hours per resource	Very low granularity: milliseconds of function execution time

price for active compute time in FaaS is typically higher than the cost of a continuously provisioned IaaS virtual machine. As a result, while FaaS is economically advantageous for bursty or intermittent workloads, it requires careful architectural consideration to avoid excessive costs in sustained, compute-intensive applications [12].

2.4.3 Object Storage

The statelessness that makes FaaS elastic also makes it incomplete. A function retains no state across invocations, so any data that must outlive a single execution has to be externalized. Such data can, in principle, travel in the payload of an invocation, but this channel is bounded, since FaaS platforms impose a limit on the size of a request that a function may receive or return, and a payload therefore cannot accommodate an artifact of arbitrary size. Nor can the functions coordinate around this limitation among themselves, since the platform assigns instances dynamically and provides no way to address a particular one [31]. Applications built on FaaS are therefore driven towards what Hellerstein et al. [31] describe as a data-shipping architecture, in which each function fetches from remote storage what it needs to compute and writes back what it produces.

The service that plays this role in commercial clouds is *object storage*. Rather than exposing a filesystem or a block device, it stores data as opaque *objects*, each identified by a key within a namespace called a *bucket*, and accessed over HTTP through an interface reachable from anywhere. The major providers all offer one: Cloud Storage on GCP [23],

Amazon S3 on AWS [4], and Blob Storage on Azure [51]. Because the size of an object is not bounded by the constraints that apply to a function payload, storage becomes the natural conduit for whatever an application must pass between its steps.

Two properties of this service govern its cost. The first is geographic: a bucket is created in a specific region, so the data it holds has a location, and an object read by a function running elsewhere crosses a regional boundary. Routing data through storage does not exempt it from the egress fees described in Section 2.3.4; it determines where those fees are incurred. Traffic between two regions is charged whether it travels in the payload of an invocation or in a request against a bucket, so what governs the expense is not the channel the data takes but the boundary it crosses, and the route by which it does so.

The second property concerns how the service is billed. Charges accrue on the volume of data held over time, on the operations performed against it, and on the traffic it generates. Storage therefore does not scale to zero in the way computation does, which is precisely the exception that Johnston admits in the definition quoted in Section 2.4.1: an object occupies space whether or not anyone reads it, and is charged accordingly. What the service does eliminate is the provisioned compute that would otherwise be needed to serve the data. A disk attached to a virtual machine is only reachable while that machine is running, and its uptime is billed regardless of whether the data is being accessed; a bucket answers a request without any instance being kept alive on the application's behalf.

2.4.4 Serverless Workflows

Storage restores to a serverless application the data that its functions cannot retain, but not the logic that sequences them. FaaS provides a unit of execution, not a unit of composition: each invocation is an isolated event, in which the platform receives a request, instantiates a function, executes it, and discards the environment. Applications, however, are rarely a single function call. They are sequences of steps, with dependencies between them, conditional branches, iterations, and failures that must be handled. Da Silva et al. [63] observe this limitation, noting that serverless platforms offer no mechanism to manage the invocations of a composite application, since they are stateless and designed for single invocations. The logic that sequences the functions must therefore reside somewhere else.

That logic is what a *workflow* expresses. A workflow describes an application as a set of interdependent tasks together with the order in which they must execute, commonly

represented as a directed acyclic graph in which nodes are computations and edges are dependencies. A *workflow management system* is the component that executes this description, automating the orchestration by managing task dependencies, data transfer, and resource allocation on behalf of the developer [63]. Such systems have a long history in scientific computing, where they coordinate large analysis pipelines, and they are traditionally deployed as persistent processes that remain alive for the duration of the application they orchestrate.

Deploying a persistent orchestrator alongside serverless functions, however, reintroduces the very inefficiency that FaaS eliminates. The orchestrator is idle whenever the functions it coordinates are computing, yet a provisioned instance is billed for that idleness. To close this gap, cloud providers offer managed *workflow engines*, which apply the serverless model to orchestration itself. The main commercial public clouds all provide one, namely AWS Step Functions [8], Google Cloud Workflows [25], and Azure Logic Apps [53]. In each, the developer declares the sequence of steps, the conditions that govern the transitions between them, and the policies that apply when a step fails, and the platform executes that declaration on demand.

What these engines provide, and what a developer would otherwise have to implement, is the machinery of reliable coordination. The execution state is checkpointed after every step and persisted by the platform, so the progress of a workflow survives the failure of any component it invokes [25]. Failed invocations are retried according to default or custom policies, and errors are handled through declarative constructs rather than application code [25]. The workflow definition itself is a machine-readable artifact, which makes the orchestration inspectable and reproducible rather than buried in an imperative script.

Taken together, these properties are what make the workflow engine a suitable substitute for a persistent orchestrator. On one hand, the engine assumes the burden of reliable coordination: the developer declares the sequence of steps and the failure policies, and the platform takes responsibility for persisting state, retrying invocations, and surviving the failure of the components it invokes, none of which needs to be implemented or maintained by the application. On the other, because the engine is itself serverless, it is billed by the steps a workflow executes rather than by the time it remains deployed, and it costs nothing while inactive or waiting [25]. An orchestration layer that coordinates an intermittent, long-running process, precisely the profile of a FL training job, therefore incurs cost only at the moments it acts, and remains economically dormant while the clients train and the models are aggregated.

The three services examined in this section compose into a complete execution model. FaaS supplies computation that scales on demand and is billed only while it runs, but which retains neither the data it operates on nor the logic that sequences it. Object storage restores the first, holding the state that outlives an invocation; a workflow engine restores the second, driving the sequence of invocations that the functions cannot coordinate among themselves. What a conventional deployment would implement as a single persistent process, holding its data in memory and running its own control loop, is thus decomposed into three managed services, none of which requires a continuously provisioned instance. It is this decomposition that Chapter 4 applies to the hierarchical federation of Section 2.2.

Chapter 3

Related Work

This chapter reviews the existing literature relevant to our work, dividing the analysis into key areas that form the foundation of our proposed solution. Initially, we discuss the intersection of FL and serverless computing, followed by an exploration of advancements in hierarchical topologies. Subsequently, we examine works that combine both serverless and hierarchical approaches. Finally, we review studies that address the challenges of globally distributed FL.

3.1 Serverless FL

To achieve scalable execution without the operational burden of maintaining continuously provisioned, often idle infrastructure, recent works have adopted the serverless computing paradigm for FL. Grafberger et al. introduced FedLess [28], a framework that executes both the client training and the aggregation as serverless functions; it demonstrated that FaaS can securely scale FL across diverse commercial cloud providers. Similarly, Kotsehub et al. developed FLoX [39], which extends FaaS capabilities to the edge; by utilizing platforms like funcX [43], FLoX enables dynamic function routing to heterogeneous edge nodes.

While the transition to serverless architectures provides benefits in terms of operational elasticity, it also introduces new economic challenges for ML workloads. Jiang et al. conducted a analysis of serverless ML training [36], revealing a performance-cost trade-off, their study demystifies the assumption that serverless computing is universally cheaper. Although the pay-per-use billing model successfully eliminates costs associated with idle infrastructure, FaaS typically incurs a higher price per unit of time than continuously provisioned IaaS instances. This trade-off establishes that while serverless can be

advantageous for environments with intermittent synchronization patterns, such as FL, it demands careful architectural optimization to ensure that the higher pricing of active compute phases does not offset the savings gained from eliminating idleness.

To minimize the overheads associated with FaaS execution and address the performance-cost trade-off, recent literature has focused on algorithmic optimizations, particularly regarding client management and straggler mitigation. In highly heterogeneous serverless environments, varying function cold starts and distinct hardware allocations cause certain clients to exhibit delayed completion times (*stragglers*), slowing down the entire global round. To tackle this, Jiang et al. introduced Pisces [37], leveraging guided asynchronous training protocols to remove the strict synchronization barriers typical of standard FedAvg, allowing faster clients to progress without waiting for stragglers. Concurrently, Elzohairy et al. proposed FedLesScan [15], which actively mitigates stragglers through adaptive client selection, dynamically scoring and filtering out slow serverless instances before they delay the aggregation process. Expanding on this concept, Chadha et al. developed Apodotiko [13], integrating a hardware-aware scoring mechanism that optimizes the selection of serverless functions based on the specific computational capabilities of the underlying nodes.

Despite these advancements in efficiency and client management, existing serverless FL frameworks still primarily operate within flat, two-tier topologies. While they successfully address computational elasticity and straggler issues, they fail to mitigate the long-haul communication bottlenecks that arise when these flat networks are dispersed globally. This limitation highlights the importance of exploring more complex network structures to further optimize communication in these environments.

3.2 Hierarchical FL (HFL)

While serverless computing environments provide the operational agility to eliminate costs associated with idle resources, addressing the communication bottlenecks inherent to highly distributed systems requires structural changes to the communication topology. HFL has emerged as a multi-tier solution to this challenge, introducing intermediate aggregation layers to alleviate the pressure on long-haul communication links. The theoretical foundation of this multi-tier approach was solidified by the HierFAVG algorithm, introduced by Liu et al. [44], which provided the convergence analysis necessary to prove the soundness of adding an intermediate aggregation layer.

Subsequent studies extended these foundations to address specific constraints of distributed networks. Wang et al. [65] provided further convergence analysis, highlighting why local edge-level aggregation effectively mitigates the impact of statistical heterogeneity by acting as a stabilizer for the global model. Research has also optimized this process through model quantization and resource-aware scheduling; Liu et al. [45] integrated quantization into the HFL framework, enabling model weight compression to reduce the communication payload between tiers. Complementarily, Liu et al. [46] explored strategies for imbalanced wireless networks, while Luo et al. [47] proposed joint resource allocation mechanisms to optimize the association between edge nodes and parameter servers, balancing energy consumption and communication latency.

While these advancements underscore the benefits of hierarchical topologies for communication efficiency, they predominantly assume traditional, server-based deployments. By relying on fixed, always-on instances, these frameworks do not account for the financial and operational limitations of persistent infrastructure.

3.3 Serverless HFL (SHFL)

Recent research has begun to explore the intersection of serverless computing and hierarchical topologies, aiming to combine the computational elasticity of FaaS with the communication efficiency of multi-tier FL. Ng et al. [59] proposed a framework that addresses client participation incentives by employing a reputation-aware coalition formation game, where edge devices dynamically form clusters to perform intermediate model updates. While they leverage serverless function invocations for these local aggregations, the framework relies on persistent, stateful reputation-management entities to govern the coalition formation. Concurrently, Hudson et al. [33] introduced FLIGHT, a FaaS-based framework that enables complex, multi-tier deployments by decoupling the control plane from the data plane. Although FLIGHT utilizes FaaS to execute training tasks, it still requires a centralized, continuously running control plane to manage the asynchronous aggregation logic and orchestrate the model flow across the hierarchy.

Despite these advancements in system flexibility and algorithmic design, existing SHFL frameworks exhibit gaps in deployment and architectural strategy. Specifically, to the best of our knowledge, they overlook the impact of geographical distribution on both performance and cost. A naive selection of the cloud region to host the global parameter server can result in increased long-haul communication latency, undermining the

efficiency gains expected from the hierarchical structure. It also disregards the network egress fees incurred whenever model parameters cross regional boundaries, which, as discussed in Section 2.3.4, vary substantially with the geographic route and can dominate the infrastructure cost of a globally dispersed federation. Furthermore, while these frameworks leverage FaaS for training tasks, they frequently rely on persistent entities, such as the stateful reputation-management entities in Ng et al. or the centralized control plane in FLIGHT, to manage the hierarchical aggregation rounds. This reliance on persistent components introduces architectural dependencies that are inconsistent with the vision of a fully serverless, pay-per-use execution model.

3.4 Globally Distributed FL

As FL scales across geo-distributed environments, geographical dispersion introduces significant communication latency and operational complexity. Recent efforts have sought to address these challenges by optimizing client-server allocation through mathematical formulations designed for static, multi-cloud Virtual Machine instances [10]. Other research focuses on communication-efficient protocols to minimize network overhead during model synchronization [18]. While these approaches successfully mitigate data transfer requirements, they rely on pre-provisioned, fixed resource locations. Consequently, these studies do not fully leverage the flexibility of cloud-native environments, where aggregation tasks can be dynamically deployed in proximity to edge nodes to minimize long-haul communication bottlenecks.

3.5 Summary and Comparative Analysis

The literature reviewed in this chapter illustrates a progression from flat, serverless-based FL frameworks to more sophisticated hierarchical topologies. However, as summarized in Table 3.1, a gap persists: to the best of our knowledge, existing SHFL frameworks still rely on persistent entities to manage the FL process orchestration. Furthermore, while hierarchical structures effectively mitigate local latency, they offer opportunities for further optimization regarding long-haul communication bottlenecks in globally dispersed networks. Specifically, these frameworks often operate without strategic regional selection, leaving both latency and network egress costs unoptimized, even though the latter varies substantially with the geographic route taken by the model parameters.

Our proposed architecture bridges these gaps by shifting from persistent, stateful controllers to a fully serverless, workflow-based orchestration strategy. Unlike prior SHFL approaches, we integrate a region-aware placement selection algorithm that optimizes the placement of the global parameter server by jointly balancing inter-continental latency, egress costs, and regional computation pricing. By aligning serverless orchestration with the underlying network topology and pricing structure, our work achieves a truly cloud-native, pay-per-use hierarchical execution model that avoids the overheads of traditional server-based deployments.

Table 3.1: Comparison of relevant works in Serverless Federated Learning and Hierarchical Federated Learning (HFL).

Work	Topology	Serverless	Orchestration	Region-aware Placement
Pisces [37]	Flat	Yes	Persistent	No
Apodotiko [13]	Flat	Yes	Persistent	No
Ng et al. [59]	Hierarchical	Partial	Persistent	No
FLIGHT [33]	Hierarchical	Partial	Persistent	No
Proposed Work	Hierarchical	Yes	Workflow-based	Yes

Chapter 4

Workflow-Based Orchestration

This chapter presents the first contribution of this work: an architecture that orchestrates a hierarchical federation without any continuously provisioned component. We begin by describing how a flat FL process maps onto a serverless workflow, and how the statelessness of the functions displaces the model into the storage layer. We then extend the model to the hierarchical case, showing how the responsibilities of a parameter server decompose into three distinct mechanisms, why the workflows that coordinate them must themselves be nested and regionally deployed, and what overheads this decomposition introduces. Finally, we detail the implementation on GCP.

Realizing the hierarchical topology of Section 2.2 on cloud infrastructure requires deciding how the training rounds it prescribes are to be coordinated. As shown in Chapter 3, the frameworks that combine HFL with serverless computing adopt FaaS for the training and the aggregation, yet retain a persistent entity to orchestrate the rounds. This retention sits uneasily with the nature of the workload, since FL training is intermittent: a round of local computation is followed by a synchronization, and between these moments the coordinating entity remains idle, while a provisioned instance is billed for that idleness regardless. The orchestration logic it runs, sequencing the invocations of each round, retrying those that fail, tracking the progress of the training, and determining when it should terminate, must moreover be implemented and maintained as application code. As established in Section 2.4.4, a managed workflow engine addresses both aspects at once: it assumes the burden of reliable coordination as a declarative service, and it is billed by the steps it executes rather than by the time it remains deployed. Our architecture builds on this to remove the last continuously provisioned component of a serverless hierarchical federation.

4.1 Serverless Orchestration Model

As illustrated in Figure 4.1, a standard FL training process can be implemented as a serverless workflow. In this model, FaaS functions (depicted in yellow) perform the core computational workloads, such as model training and aggregation, while the workflow engine (depicted in blue) manages the execution state, iteration counters, and termination logic. The process follows an iterative cycle: in each round, the workflow orchestrates participant selection and local training, followed by a global aggregation step. As depicted by the blue decision diamond in the figure, this execution loops continuously until a specific termination criterion is met, such as reaching a target model accuracy (e.g., $\text{Acc.} \geq \text{Target}$), completing a maximum number of rounds, or exhausting a predefined time budget. This approach decouples the training loop from the underlying infrastructure, enabling a fully pay-per-use execution.

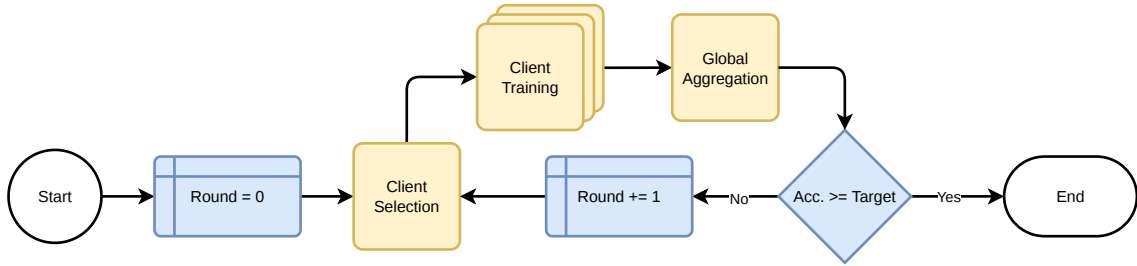


Figure 4.1: Example of a flat Federated Learning (FL) process implemented as a serverless workflow. Yellow boxes represent stateless functions, while blue boxes denote control-flow steps managed by the workflow engine. The decision diamond evaluates the termination criterion, here the accuracy (Acc.) of the global model against a target value.

A constraint of this model is the statelessness of the functions that compose it. As discussed in Section 2.4.2, a FaaS instance retains no persistent local state across invocations, so any data that must survive between steps has to be externalized. In our architecture, this data is the model itself: rather than being passed through the payloads exchanged by the workflow engine, the model parameters are persisted in a storage bucket, and the functions exchange only the storage references needed to locate them. A training function therefore begins by retrieving the current model from storage and ends by writing its updated weights back, while an aggregation function retrieves the set of models it must consolidate and writes the resulting one.

This indirection through storage has three consequences that shape the rest of this work. Operationally, it keeps the orchestration layer lightweight, since the workflow engine routes references rather than multi-megabyte models, and it renders the functions freely

relocatable, as any instance can resume the process from the persisted state. It also makes the execution robust to failure: because every invocation reconstructs its state from the storage layer rather than from the residue of a previous attempt, a function that fails midway can simply be retried by the workflow engine, with no compensating logic in the application. Financially, however, it means that the model traffic of the federation flows through the storage layer rather than through the function invocations, so it is the reads and writes against these buckets that cross regional boundaries and incur the egress fees described in Section 2.3.4.

4.2 Hierarchical Coordination Strategy

However, scaling this model to a hierarchical architecture necessitates a more sophisticated coordination strategy. In an HFL setting, relying on a single workflow becomes impractical, as the system must manage two distinct aggregation cycles: frequent regional updates and periodic global synchronizations. To address this, our proposed architecture implements a nested orchestration logic, as shown in Figure 4.2.

In this architecture, the responsibilities that a traditional deployment concentrates in a single persistent process are separated into the three mechanisms described in Section 2.4. Orchestration, that is, sequencing the invocations of each round, tracking their progress, and determining when training terminates, is handled by the workflow engine; computation, which comprises the local training, the selection of participants, and the aggregation defined by Equations 2.6 and 2.7, is performed by stateless functions; and the model state itself resides in the storage layer.

Nesting the workflows, rather than centralizing the orchestration in a single one, is not merely a matter of modularity: it is what allows the coordination itself to be geographically distributed. Each *edge workflow* is deployed in the cloud region of the edge server it represents, alongside the functions and the storage bucket it operates on, so the invocations it issues for participant selection, local training, and regional aggregation remain within that region. Were a single workflow to coordinate every client directly, each of the κ_2 regional rounds would require invocations travelling from the global region to every client, reintroducing the long-haul exchanges that the hierarchical topology is meant to confine to the global synchronization step. Nesting the workflows keeps the frequent regional coordination on intra-regional links and reserves the inter-regional path for the exchange between the *global workflow* and the edge workflows.

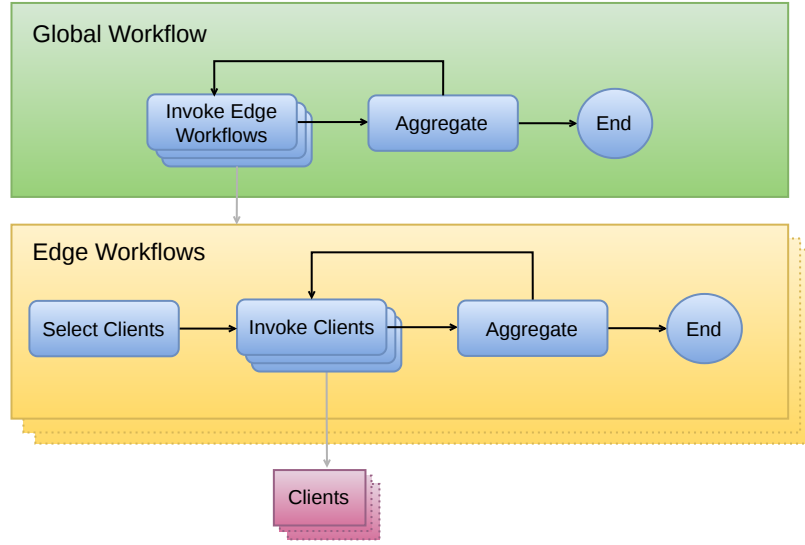


Figure 4.2: Proposed hierarchical workflow orchestration. The global workflow iterates over global rounds, invoking the edge workflows in parallel and aggregating the regional models they return. Each edge workflow, deployed in the region of the edge server it represents, iterates over the κ_2 regional rounds, selecting the participating clients, invoking the training function on each, and consolidating their updates into a regional model.

This nested structure maps directly onto the HierFAVG procedure of Algorithm 1. The outer loop over global rounds is realized by the global workflow, which at each iteration invokes the edge workflows in parallel and, upon their completion, triggers the global aggregation, repeating this cycle until the configured termination criterion is met. The inner loop over the κ_2 regional rounds is realized by each edge workflow, which iterates client selection, local training, and regional aggregation before returning its consolidated regional model. The parameter κ_1 , in contrast, is not a control-flow construct at all: it is an execution parameter passed to the training function, which performs the corresponding number of local updates within a single invocation. The three levels of the algorithm thus correspond to three distinct mechanisms, namely the global loop to the outer workflow, the regional loop to the nested workflow, and the local updates to the internal logic of a stateless function.

This distinction carries a cost that constrains the choice of κ_1 and κ_2 . Only the first two levels generate billable state transitions in the orchestration layer, so increasing κ_1 lengthens a single function invocation without adding any orchestration overhead. The parameter κ_2 , in contrast, multiplies both: each regional round triggers a fresh set of client invocations followed by an aggregation, so the number of function instantiations grows with it, and with them the cold starts described in Section 2.4.2, as does the number

of workflow transitions the engine bills. Increasing the frequency of regional aggregation therefore reduces the long-haul traffic at the price of a larger number of function instantiations and workflow transitions, and this tension between communication savings and orchestration overhead is quantified empirically in Chapter 6, which shows that the hierarchical topology becomes advantageous only once the accumulated communication savings surpass these overheads.

The proposed architecture consists of three core stateless functions, coordinated by the serverless workflow engine. These components are modular, receiving execution-specific parameters that define their behavior in each round:

- **Training Function:** Executed by clients, this function performs local model updates. As input, it receives execution parameters such as the current round number, training hyperparameters, and the number of local steps (κ_1). It retrieves the current model parameters from a storage bucket, executes the local training process, and outputs the updated weights back to the storage layer, alongside metadata such as the number of processed examples, ensuring that the execution state remains fully decoupled.
- **Aggregation Function:** This component manages model averaging for both edge and global tiers. As input, it receives the selected aggregation strategy and the storage references of the targeted model updates. It retrieves these individual models from the storage bucket, applies the configured aggregation algorithm, and evaluates the resulting model's performance. Finally, it outputs the consolidated model alongside its evaluation metrics back to the storage layer. Because the same function serves both tiers, differing only in the set of models it consolidates, the two aggregation rules of Equations 2.6 and 2.7 are realized by a single deployable component.
- **Client Selection Function:** Invoked by edge workflows, this component manages participant selection for each training round. It takes the current execution configuration and round number as input, consults the registry of clients associated with its region, and applies the defined selection strategy. It outputs the list of selected participants.

4.3 Implementation

The entire system is implemented on the public commercial cloud GCP [26]. The functional components are deployed as Python-based Cloud Run Functions [20], while orchestration is managed by GCP Workflows [25] and model parameters are persisted in Cloud Storage [23] buckets. Consistent with the coordination strategy described in Section 4.2, the global workflow is deployed in the region selected for the global parameter server, while each edge workflow, together with the functions and storage bucket it operates on, is deployed in the region of its corresponding edge server.

For asynchronous signaling between workflows, we utilize Google Cloud Pub/Sub [24]. As a managed publish-subscribe message broker, this type of service allows independent distributed components to communicate asynchronously; producers publish events to central topics without needing to establish direct, point-to-point connections with the consuming services. Specifically, in our architecture, these messages transmit execution events, such as signaling the completion of regional training and broadcasting the start of a new global round. This service facilitates cross-region coordination by decoupling the execution of different workflow tiers, allowing for message delivery latencies on the order of 100 milliseconds on a global scale [24].

The deployment of the cloud infrastructure is automated using Terraform [30], adopting an Infrastructure-as-Code (IaC) paradigm. By defining cloud resources through declarative configuration files, IaC eliminates the manual and error-prone processes traditionally associated with infrastructure provisioning [58]. This approach is important for large-scale distributed systems, as it ensures environmental consistency and reproducibility across multiple regions. In our case, the same declarative definition is instantiated once per participating region, which is what makes it practical to deploy and tear down a federation spanning four continents for each experimental run. Furthermore, treating infrastructure as version-controlled software artifacts facilitates rapid tear-down and redeployment, which is important for validating experimental results and ensuring the reproducibility of the proposed serverless architecture.

This IaC approach, combined with the utilization of provider-independent service patterns, ensures that the hierarchical orchestration remains deployable in diverse cloud environments without necessitating structural modifications to the architecture. Specifically, the architecture achieves this compatibility by relying on managed services that are supported by major cloud vendors. To illustrate this, Table 4.1 summarizes the equiva-

lent managed services across GCP [26], AWS [6] and Azure [56] for the core functional components and coordination layers.

Table 4.1: Equivalent managed services across GCP, AWS, and Azure, illustrating the cross-cloud portability of the proposed serverless stack.

Component	GCP	AWS	Azure
Functions	Cloud Run Functions [20]	AWS Lambda [7]	Azure Functions [52]
Orchestration	GCP Workflows [25]	Step Functions [8]	Logic Apps [53]
Storage	Cloud Storage [23]	Amazon S3 [4]	Blob Storage [51]
Signaling	Pub/Sub [24]	SNS [5]	Service Bus [54]

Chapter 5

Global Server Placement

The hierarchical topology confines the most frequent exchanges to intra-regional links, but it does not eliminate the long-haul path: at every global round, each edge server must still reach the global parameter server. Where that server is placed therefore determines the cost and the latency of the one exchange the hierarchy cannot localize. As established in Section 2.3.4, neither of these quantities is uniform across regions, so this placement is a design decision rather than a matter of convenience.

Our region selection method is based on a scoring function that evaluates candidate regions against three variables: the network egress rate, the communication latency, and the computation cost. Although the values used to instantiate these variables are obtained from GCP pricing tables and latency measurements, the scoring logic itself is provider-independent and applicable to any platform with a comparable regional structure. Costs associated with orchestration and messaging services are excluded from the model, as they are billed uniformly regardless of the deployed region and therefore cannot discriminate between candidates.

5.1 Evaluation Metrics

Each candidate region is characterized by three metrics, corresponding to the three dimensions along which regions differ: the price of moving model parameters to and from that region, the time such transfers take, and the price of executing the aggregation there. The first two capture the network, and the third the compute. Together they express the tension the placement must resolve, since the region that minimizes one is not necessarily the region that minimizes the others.

5.1.1 Network egress rate ($\mathcal{E}$)

This variable represents the rate at which the transfer of model parameters between the global parameter server and the edge parameter servers is charged, an exchange that occurs at every global round. As established in Section 2.3.4, cloud providers charge these transfers by volume, and the applicable rate depends on the geographic route the data follows, so the same synchronization is billed differently depending on where the global server resides. The volume itself, however, is the same for every candidate, since it is determined by the size of the model and the number of edge servers rather than by the location of the global one. What distinguishes the candidates is therefore the rate alone, and it is the rate that $\mathcal{E}$ expresses, in US dollars per GiB. To obtain it, we take the inter-region transfer rates published in the GCP network pricing documentation [27] for the routes connecting a candidate region to each of the edge regions.

The rates we collected vary with the route: transfers from North American regions to Europe, for instance, are considerably cheaper than those targeting South America, which can cost up to three times as much for the same volume of data. This makes egress the most volatile of the three metrics, and the one that most strongly differentiates candidate regions.

5.1.2 Communication latency ($\mathcal{L}$)

This variable represents the time taken to transfer the model parameters between the global parameter server and the edge parameter servers. As described in Section 4.1, the model never travels in the payload of a function invocation: it is written to and read from a storage bucket, so what the system experiences at every global round is the latency of a function retrieving an object from a bucket in another region. The measurement mirrors this exactly. A FaaS metering function was deployed across the candidate source regions and instructed to download a 1 MB test payload from storage buckets located in each of the target regions. To account for transient fluctuations, each measurement was repeated five times and the average was taken.

The measurements reveal a variation among intercontinental connections, with latencies ranging from approximately 600 ms (e.g., between the United States and Europe) to over 1500 ms (e.g., between South America and Australia). The communication overhead can therefore vary by more than a factor of 2.5 depending solely on the geographic placement of the global server, which is what makes $\mathcal{L}$ a meaningful discriminator among

candidate regions.

5.1.3 Computation cost ($\mathcal{P}$)

This variable represents the direct expense of executing the stateless functions within the serverless environment. FaaS platforms calculate this cost based on the allocation of compute resources multiplied by the total execution duration. Specifically, the billing model accounts for a combination of memory (typically billed in gigabyte-seconds, GB-s) and processing power (billed in vCPU-seconds), and may also include hardware accelerators such as GPUs depending on the workload and provider. Crucially, the baseline unit prices for these allocated resources are not uniform globally; they fluctuate depending on the specific geographical region where the function is deployed. To calculate $\mathcal{P}$, we derive the costs directly from the GCP Cloud Run pricing tables [21], applying the regional rates corresponding to the exact memory and vCPU tiers required.

The GCP pricing tables place each region in one of two tiers, so a candidate is either among the cheaper ones or among the more expensive ones, the latter costing approximately 40% more per second of execution. This makes $\mathcal{P}$ the coarsest of the three metrics, since it separates the candidates into two groups rather than ordering them along a continuum, as network egress and communication latency do. It remains, nonetheless, an important factor in modeling the overall economic efficiency of the placement, and one that can decide between candidates that the other two metrics rank as equivalent.

5.2 Region Selection Score

Based on these variables, we propose a region selection score to evaluate each candidate region $r \in \mathcal{R}$, where $\mathcal{R}$ represents the set of all available cloud regions. The scoring function is defined as:

$$Score(r) = \alpha \cdot \text{Norm} \left(\sum_{e=1}^L \mathcal{E}^{(e)}(r) \right) + \beta \cdot \text{Norm} \left(\sum_{e=1}^L \mathcal{L}^{(e)}(r) \right) + \gamma \cdot \text{Norm}(\mathcal{P}(r)) \quad \forall r \in \mathcal{R} \quad (5.1)$$

In (5.1), L is the number of active edge parameter servers. The terms $\mathcal{E}^{(e)}(r)$ and $\mathcal{L}^{(e)}(r)$ denote, respectively, the egress rate and the empirical communication latency for the route between the candidate global region r and the region hosting the edge server e .

The term $\mathcal{P}(r)$ represents the computation cost within region r .

Because these metrics possess distinct units and magnitudes, the function $\text{Norm}()$ applies a min-max normalization to scale each feature strictly into the $[0, 1]$ interval. For any given metric value x , the normalized value is computed as:

$$\text{Norm}(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (5.2)$$

In (5.2), $x_{\min}$ and $x_{\max}$ represent, respectively, the absolute minimum and maximum values of the evaluated metric computed across the entire set of available candidate regions R . The weights α , β , and γ (where $\alpha + \beta + \gamma = 1$) are tunable parameters that prioritize either financial cost reduction or performance optimization. The region r that yields the minimum overall score is selected as the optimized location for the global parameter server.

Chapter 6

Results

This chapter evaluates the proposed architecture along the two dimensions that motivate it: the efficiency of the hierarchical topology under serverless execution, and the impact of the region selected to host the global parameter server. The evaluation combines two complementary approaches. We first describe the experimental setup, detailing the workloads, the data partitioning, and the cloud configurations under which the system was deployed. We then present the results of real deployments on GCP, which measure the behaviour of the architecture under actual network and infrastructure conditions, culminating in a world-scale case study distributed across four continents. Finally, because the operational and financial constraints of cloud deployment limit how far such experiments can be scaled, we present a simulator calibrated with data collected from these executions, through which the analysis is extended to a federation of thousands of clients.

6.1 Experimental Validation

To validate the proposed architecture in a realistic, globally distributed environment, we conduct a series of experiments on GCP. Our evaluation is structured to address three research questions:

1. **Communication Efficiency:** Does confining the frequent exchanges to regional links shorten the duration of a training round, or is the saving in long-haul communication consumed by the orchestration and cold start overheads that the edge aggregation layer introduces?
2. **Time-to-Accuracy:** Does a shorter round translate into a shorter training run, once the number of rounds required to reach the target accuracy is taken into account?

3. **Placement Optimization:** To what extent does the region selection algorithm improve overall efficiency by optimizing the trade-off between communication performance and operational costs?

6.1.1 Experimental Setup

All tests were conducted on GCP using the serverless components detailed in Section 4.3. While the cloud provider offers more regions, we restrict these experiments to six candidate locations: Asia, Australia, Europe, North America, South America, and the United States. Other available regions are disregarded as candidates for the global server placement to maintain manageable experimental complexity.

6.1.1.1 Datasets and Models

Three workloads are used. The first two are standard image classification benchmarks, long established in the ML literature and adopted by the FL works that ground this dissertation, which makes them suitable for validating the architecture. MNIST [41] consists of 28×28 grayscale images of handwritten digits, to be classified into ten classes, and is the simpler of the two, both in the size of its samples and in the size of the model it requires. CIFAR-10 [40] consists of 32×32 colour images drawn from ten everyday object categories, and poses a substantially harder task, demanding a deeper model and considerably more training to reach comparable accuracy.

The models trained on them follow this progression. For MNIST, we employ a convolutional network with two convolutional layers followed by two fully connected layers, totalling 21,840 trainable parameters. For CIFAR-10, we employ a deeper network organized into three convolutional blocks, each combining convolutional layers with batch normalization and max pooling, followed by a fully connected head, amounting to 5.85 million parameters. Both architectures reproduce those adopted by Liu et al. [44] in the original HierFAVG evaluation. The two workloads thus differ by more than two orders of magnitude in model size, which is precisely what makes them complementary here: since the model is what traverses the network at every synchronization, the volume of data exchanged per round differs accordingly, and with it the relative weight of communication against computation.

The third workload departs from these benchmarks in that it is inherently geodistributed. So2Sat LCZ42 [69] is a geospatial dataset built from satellite imagery of

cities across the world, in which each patch is labelled with its local climate zone, a classification of urban land use. Its samples are co-registered patches acquired by two distinct sensors: Sentinel-1, which provides eight channels of synthetic aperture radar data, and Sentinel-2, which provides ten multispectral channels. We restrict the task to ten urban land-use classes, yielding 174,746 training and 43,500 testing patches of 32×32 pixels. What makes this dataset valuable for our purposes is that its samples carry a real geographic origin: each city is a natural client, situated in an actual location on the globe, so the federation it induces is not an artificial partition of a centralized dataset but a genuinely dispersed one, with the statistical heterogeneity that follows from cities in different parts of the world being built differently.

Because the two sensors capture complementary physical properties of the same scene, the model is organized as a dual-branch network: two parallel convolutional towers, each mirroring the block structure used for CIFAR-10, extract features independently from the Sentinel-1 and Sentinel-2 inputs, and their outputs are concatenated before a shared fully connected classification head. The resulting model comprises 11.2 million parameters, making it the largest of the three.

6.1.1.2 Data Partitioning

The MNIST and CIFAR-10 experiments adopt an IID partition, in which the training samples are shuffled and distributed uniformly among the clients, so that every local dataset is a representative sample of the global distribution. This isolates the architectural effects under study from the influence of statistical heterogeneity.

The So2Sat case study, in contrast, adopts a non-IID partition, reflecting the fact that cities in different regions of the world exhibit markedly different distributions of urban land-use classes. We generate this partition using a Dirichlet distribution [32], the standard mechanism for inducing controlled label skew in FL. For each class c , a vector of proportions $p_c = (p_{c,1}, \dots, p_{c,N})$ is drawn from a symmetric Dirichlet distribution $\text{Dir}_N(\alpha_{Dir})$, and client i receives a fraction $p_{c,i}$ of the samples belonging to class c . The concentration parameter α_{Dir} governs the severity of the resulting skew: as it grows, the drawn proportions approach uniformity and the partition converges to the IID case, whereas as it approaches zero, the mass concentrates on a few clients per class, so that each client holds samples of only a handful of classes. We adopt $\alpha_{Dir} = 0.5$, a value widely used in the FL literature, which yields a pronounced but not degenerate skew: local class distributions differ substantially across clients, yet no client is restricted to a single class.

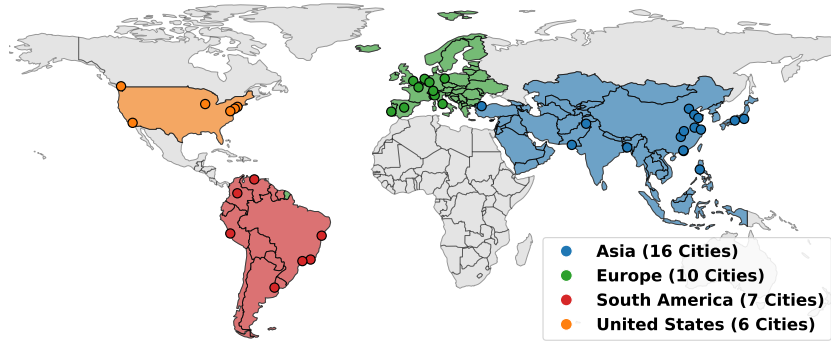


Figure 6.1: Geographical distribution of the 39 cities selected from the So2Sat dataset used in the Global Multi-Region experiment. In our architecture, clients representing these cities are mapped to and orchestrated by regional workflows in their nearest Google Cloud regions (Asia, Europe, South America, and United States).

6.1.1.3 Regional Setup and Resource Allocation

The MNIST and CIFAR-10 benchmarks serve as an initial architectural validation and adopt a dual-region configuration, with clients distributed between South America (southamerica-east1) and the United States (us-central1). For these workloads, aggregation functions were provisioned with 16 GB of memory and 6 vCPUs, while training functions received 8 GB of memory and 4 vCPUs.

The So2Sat case study, in turn, evaluates the system in a world-scale deployment. Its 39 representative cities are modeled as individual clients and mapped to their geographically nearest cloud region, resulting in four distinct edge workflows: Asia (asia-east1, 16 clients), Europe (europe-north2, 10 clients), South America (southamerica-east1, 7 clients), and the United States (us-central1, 6 clients). While Figure 6.1 depicts the physical coordinates of the cities, the client functions were effectively deployed within these mapped cloud regions to simulate regional processing. For this workload, aggregation functions were provisioned with 32 GB of memory and 8 vCPUs, while training functions were allocated 12 GB of memory and 4 vCPUs.

6.1.2 Communication Efficiency

This subsection evaluates the hierarchical model against a flat baseline on the MNIST and CIFAR-10 benchmarks, addressing the first research question. What is at stake is whether the reduction in long-haul communication is large enough to absorb the overheads the edge aggregation layer introduces, namely the additional cold starts and workflow transitions of the serverless execution.

To evaluate this efficiency, we compare the total wall-clock time for a fixed set of aggregation steps. Following the Dual-Region configuration described at 6.1.1.3, these experiments involved 20 clients evenly distributed across two regions. The global workflow was deployed in the Europe region (`europa-north2`) to introduce significant inter-continental latency. Furthermore, the reported results represent the average of three independent runs, exhibiting a coefficient of variation of approximately 8% across all individual executions.

In both configurations, the computational workload at the client level remains identical, with the number of local updates (κ_1) fixed at 6. The distinction between the two architectures lies in the aggregation path: the flat baseline synchronizes clients directly with the global server (k times), whereas the hierarchical model performs κ_2 regional aggregation steps at the edge before a global synchronization.

The average wall-clock times are visualized in Figure 6.2. For a small number of rounds (k or $\kappa_2 = 1$), the hierarchical model introduces significant overhead for both datasets due to its architectural complexity and the additional intermediate aggregation step. However, as the communication savings accumulate, the hierarchical model becomes more efficient, eventually surpassing the flat baseline.

As illustrated, the benefits for the smaller MNIST model are already apparent at $k_2 = 5$, where the hierarchical setup achieves a reduction of 10.3% compared to the flat baseline. By $k_2 = 10$, this advantage increases to a 25.4% reduction in total wall-clock time. In contrast, for the more computationally intensive CIFAR-10 model, the performance at $k_2 = 5$ is nearly identical across both configurations. The hierarchical advantage for CIFAR-10 only becomes evident at $k_2 = 10$, yielding a 10.6% improvement. This suggests that for larger models, the longer computation time per round initially masks the communication savings, requiring a greater number of rounds for the cumulative benefits of the hierarchical structure to become dominant.

6.1.3 Time-to-Accuracy

While the previous analysis demonstrated per-round efficiency, this experiment evaluates end-to-end performance by measuring the total wall-clock time required to reach a target model accuracy. Accuracy is computed by validating the updated global model against a hold-out test dataset at the global workflow level after every global round. We compare hierarchical configurations against the flat baseline following the Dual-Region configuration, these experiments involved 100 clients evenly distributed across two regions. Consistent

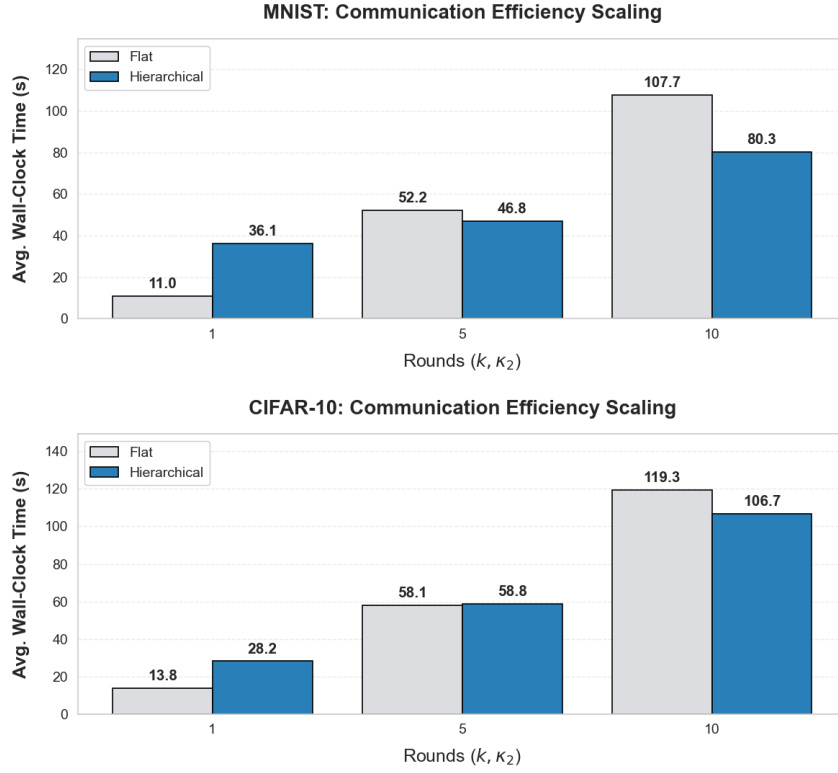


Figure 6.2: Average wall-clock time comparison between the flat and the Hierarchical Federated Learning (HFL) configurations, for the MNIST and CIFAR-10 datasets, across different numbers of aggregation rounds.

with the setup used in the previous experiments, the global workflow was deployed in the Europe region (`eu-north-2`) to induce significant inter-continental latency. The reported results represent the average of three independent runs; across the totality of these individual executions, we observed a mean coefficient of variation of approximately 10%.

The convergence results, summarized in Figure 6.3, reveal that all tested hierarchical configurations for MNIST outperformed the flat baseline. The; setup ($\kappa_1 = 20, \kappa_2 = 3$) reduced total training time by 32%, saving over 150 seconds. Although the highly hierarchical MNIST configuration ($\kappa_1 = 10, \kappa_2 = 6$) minimized global rounds to just 3, it introduced cumulative overheads that made it slower than the best setup.

Similar trends were observed for the CIFAR-10 dataset, where the configuration ($\kappa_1 = 25, \kappa_2 = 2$) reduced execution time by 24.4% and global rounds from 23 to 18. However, the CIFAR-10 experiment also highlights the risks of excessive hierarchy: the ($\kappa_1 = 10, \kappa_2 = 5$) configuration caused a massive spike in wall-clock time. This performance degradation occurs because the increased frequency of regional aggregations, combined with the larger model size of CIFAR-10, results in an overhead that far outweighs any

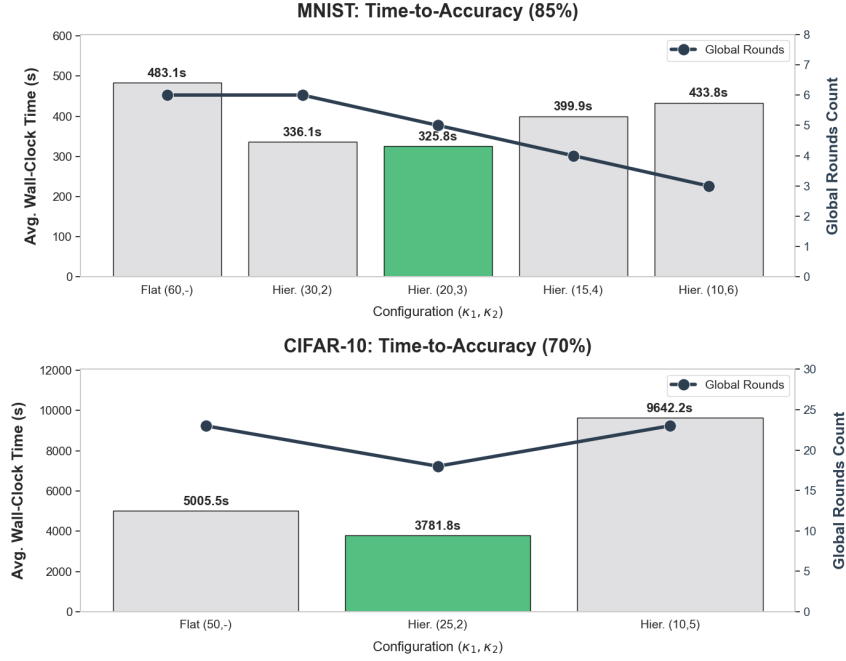


Figure 6.3: Performance trade-offs to reach the target accuracy for MNIST and CIFAR-10, comparing the flat baseline against hierarchical configurations with different values of κ_1 and κ_2 . Bars represent the total wall-clock time (left axis), while the line tracks the number of global rounds required for convergence (right axis).

gains in network latency reduction.

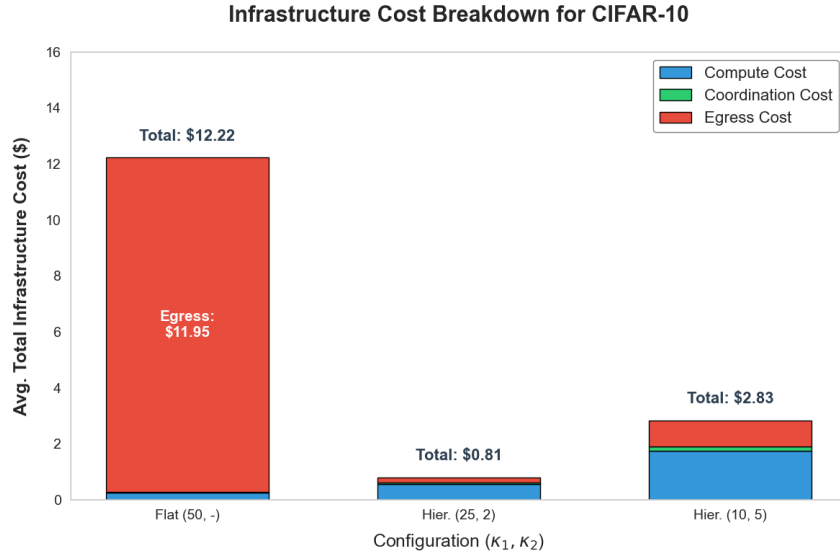


Figure 6.4: Infrastructure cost breakdown for CIFAR-10 reaching 70% accuracy. The comparison highlights the impact of hierarchical topology on network egress, aggregation function computation, and orchestration costs.

As visualized in Figure 6.4, the infrastructure cost for the best CIFAR-10 setup dropped by 93.4%, falling from \$12.22 to \$0.81. This was primarily driven by a 98.4% reduction in network egress costs (\$0.19 vs. \$11.95), achieved by confining most traffic

within regional boundaries.

Furthermore, our workflow-based orchestration proved more efficient than the traditional approach. Orchestration cost approximately \$0.05, whereas the persistent coordination entity on which the SHFL literature commonly relies would incur roughly \$0.10 to keep even the most economical general-purpose IaaS instance (**c4-standard-2**) alive throughout the execution, half of which is paid while the entity sits idle. In absolute terms the difference is a few cents, since this benchmark converges in under an hour. What the comparison establishes, however, is a ratio rather than an amount: the persistent orchestrator is billed for the entire wall-clock duration of the training, whereas the workflow engine is billed only for the state transitions it performs, a quantity that grows with the number of rounds and not with their length.

6.1.4 Placement Optimization and Global Case Study

This subsection evaluates the impact of our proposed region selection algorithm in a truly global scenario using the So2Sat dataset. While the hierarchical topology provides the structural foundation for efficiency, the physical location of the global server remains an important architectural choice for geographically dispersed FL.

The comparison requires a baseline, and here the literature offers none: as discussed in Chapter 3, to the best of our knowledge no existing SHFL framework addresses the placement of the global tier, so there is no competing method against which to measure ours. The relevant comparison is therefore not against another algorithm, but against the decision a practitioner would take in its absence. That decision is geographic: presented with a map of the participating regions, the natural choice is the one that appears most central, minimizing the apparent distance to the most distant edge servers. We adopt this as our baseline and refer to it as the *intuitive* placement, and its interest lies precisely in how reasonable it seems. Following the client distribution and regional mapping described in Section 6.1.1.3, this criterion selects **europe-north2**, which, as Figure 6.1 illustrates, sits between the clusters in Asia, South America, North America, and Europe and thus appears to be the most balanced placement available.

What the intuitive choice cannot see, however, is that geographic distance is only one of the quantities that a placement determines, and not the one that varies most. As established in Chapter 5, the latency of a route is not a linear function of the distance it spans, egress rates differ by as much as a factor of three between routes of comparable length, and the price of computation varies across regions by a further twenty percent.

The scoring function of Equation 5.1 weighs these three quantities against one another, and the region it selects need not be the one a map would suggest.

Applying the scoring function of Section 5.2 across all candidate regions bears this out. Evaluating the permutations of the weights $\{0.4, 0.3, 0.3\}$ for α , β , and γ , and thereby shifting the priority between latency, egress, and computation costs, the algorithm consistently identified **us-central1** as the optimized region. This stability indicates that the selection does not hinge on a particular weighting: the region offers a superior trade-off regardless of which of the three criteria is prioritized.

To assess the impact of our contributions, we compared three configurations reaching a target accuracy of 50%: (i) **Flat Baseline**, where all 39 clients communicate directly with a single global server in Europe (**eu-north2**); (ii) **HFL Intuitive**, a hierarchical architecture ($\kappa_1 = 25, \kappa_2 = 2$) with the global server in Europe; and (iii) **HFL Optimized**, a hierarchical setup with the global server in the United States (**us-central1**). Each configuration represents the average of three independent runs, exhibiting a mean coefficient of variation of approximately 7% across the scenarios.

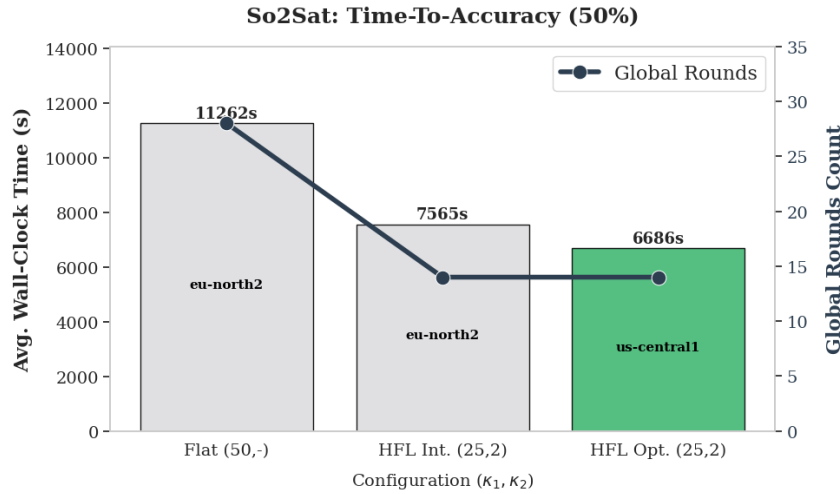


Figure 6.5: Performance trade-offs to reach the target accuracy for So2Sat, comparing the flat baseline against the intuitive and the optimized hierarchical placements of the global parameter server. Bars represent the total wall-clock time (left axis), while the line tracks the number of global rounds required for convergence (right axis).

The results, visualized in Figures 6.5 and 6.6, demonstrate that the transition from *Flat* to *HFL (Intuitive)* yielded substantial gains: total execution time dropped by nearly 33%, and infrastructure costs were reduced by 86.5%. These findings reinforce the performance trends observed in the previous analysis, confirming that the hierarchical topology effectively mitigates the bottlenecks of long-haul data transfers even when scaling to a world-wide scenario.

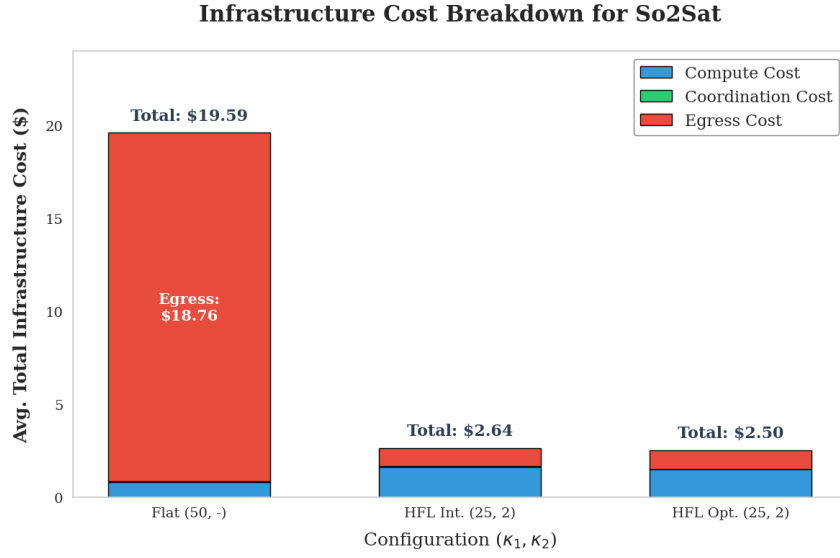


Figure 6.6: Infrastructure cost breakdown for the So2Sat case study. The chart highlights the reduction in network egress costs achieved by the hierarchical architecture and the minimal overhead of serverless coordination.

Second, the transition from intuitive to optimized placement unlocked further efficiency. Deploying the global server in **us-central1** reduced the execution time by an additional 11.6%, achieving a 40.6% total improvement over the baseline, and reduced the total infrastructure cost by 5.3%. Crucially, both regions share identical unit pricing for computation and network egress between edge workflow regions; the gains were driven solely by our algorithm identifying that **us-central1** offered superior connectivity, with average latencies 33.8% lower than **europa-north2**.

As visualized in Fig. 6.6, coordination costs remain negligible (\$0.03), reinforcing the efficiency established in previous experiment. In contrast, the persistent coordination common in existing SHFL literature would incur \$0.18 for the equivalent runtime on the most economical general-purpose instance available (**c4-standard-2**).

6.2 Simulated Validation

The empirical evaluation of the previous section is bounded by the operational and financial limits of deploying a real federation on a commercial cloud. To extend the analysis beyond these limits, we developed a simulator focused on the interplay between the FL architecture and network communication performance. This section first describes how the simulator models wall-clock time and infrastructure cost, then validates it against the measurements obtained from the real deployments, and finally uses it to evaluate the

architecture at a scale of thousands of clients.

Under a *same-effort* principle, the methodology isolates infrastructure efficiency by measuring costs and wall-clock time over an equivalent number of rounds. The simulator maps the logical phases of the FL lifecycle to the physical constraints of a geo-distributed infrastructure.

6.2.1 Wall-Clock Time Modeling

The simulator estimates wall-clock time by decomposing a global round into computational processing (t_{proc}) and network communication (t_{comm}). The communication time between any two regions a and b follows a simplified model of the TCP square root law [60], as defined in Eq. 6.1:

$$t_{comm}(a, b) = \frac{S_{mod}}{BW \cdot \eta(a, b)} + \mathcal{L}(a, b) \quad (6.1)$$

$$\text{with } \eta = \min \left(1, \sqrt{\frac{\mathcal{L}_{base}}{\mathcal{L}(a, b)}} \right)$$

where S_{mod} is the model size, BW the nominal bandwidth, and $\mathcal{L}(a, b)$ is the inter-region latency. The factor η penalizes throughput as latency exceeds the intra-region baseline ($\mathcal{L}_{base}$). This approach abstracts complex parameters, such as packet loss rates and congestion window, into a latency-dependent efficiency ratio. Such simplification is justified as the primary goal of this evaluation is to analyze the relative impact of network topology and orchestration paradigms on global-scale FL, rather than performing microscopic protocol modeling.

The computational times t_{proc}^{client} , t_{proc}^{edge} , and t_{proc}^{global} represent the processing times for local training, edge aggregation, and global aggregation, respectively. These are derived from the empirical averages observed in our physical experiments. Although these values may vary at larger scales, using empirical constants serves as a reasonable estimation to isolate the impact of network topology and infrastructure costs. Following the notation of Section 2.2, let N denote the number of clients, L the number of edge servers, and C_e the set of clients associated with the edge server $e \in \{1, \dots, L\}$. The total round time for the flat (T_{round}^{flat}) and hierarchical (T_{round}^{hier}) topologies is then modeled in Eq. 6.2:

$$\begin{aligned}
T_{round}^{flat} &= \max_{i \in \{1, \dots, N\}} (2 \cdot t_{comm}(g, i) + t_{proc}^{client}) + t_{proc}^{global} \\
T_{round}^{hier} &= \max_{e \in \{1, \dots, L\}} (2 \cdot t_{comm}(g, e) + T_{edge}(e)) + t_{proc}^{global} \\
&\text{where} \\
T_{edge}(e) &= \kappa_2 \cdot \left(\max_{i \in C_e} (2 \cdot t_{comm}(e, i) + t_{proc}^{client}) + t_{proc}^{edge} \right)
\end{aligned} \tag{6.2}$$

In these formulations, g denotes the region hosting the global parameter server, and t_{comm} is evaluated between the regions in which the corresponding entities reside. The term $T_{edge}(e)$ is central to the hierarchical advantage, as it encapsulates the entire sub-process of regional training and aggregation. Since these operations occur in parallel across all edge servers, the global system is not penalized by the total sum of client latencies, but only by the bottleneck of the slowest one. Furthermore, by keeping the high-volume data exchange between an edge server and its clients within the same region, $T_{edge}(e)$ benefits from the intra-region baseline $\mathcal{L}_{base}$, maximizing the effective bandwidth during the most frequent phases of the FL process.

This decomposition reveals how the hierarchical architecture mitigates the impact of transcontinental delays. While the flat model forces every global round to wait for the slowest link among all N clients, the hierarchical model restricts this penalty to the L edge servers, and only at the infrequent global synchronization. By decoupling the high-frequency local training from the long-haul global backbone, the system can scale the number of clients within each C_e with minimal impact on the total round time.

6.2.2 Infrastructure Cost Modeling

The financial impact of each paradigm is evaluated by modeling the infrastructure costs. In both the *always-on* and *serverless* models, the total cost is a composition of three primary factors: (i) the computation cost of the aggregation functions; (ii) the orchestration cost, representing the expenses for managing the learning process; and (iii) the network egress fees incurred during inter-regional model transfers.

The fundamental distinction between the two paradigms lies in the computation cost component. For always-on architectures, we consider the pricing of the smallest general-purpose instances providing computational resources (CPU and memory) equivalent to the FaaS environment. In this persistent model, computation cost is determined by the

continuous allocation of resources; since instances remain active across all execution time, the expense is proportional to the cumulative wall-clock time across all global rounds.

In contrast, the serverless model decouples billing from idle periods by charging strictly for the active execution time of the aggregation functions. While FaaS typically carries a higher price per unit of time compared to provisioned instances, this pay-per-use approach allows for a direct quantification of the financial benefits of the serverless model.

6.2.3 Simulator Validation

To ensure the reliability of the simulator, we validated it by directly replicating the placement optimization and global case study described in Section 6.1.4. By using the exact same 39-client configuration and regional distribution, we calibrated the model with the empirical computational processing speeds and the global inter-region latency map ($\mathcal{L}$) previously established in Section 5.2. This approach allows for a direct comparison between real-world GCP execution and simulated results, ensuring that the results are consistent with physical infrastructure performance.

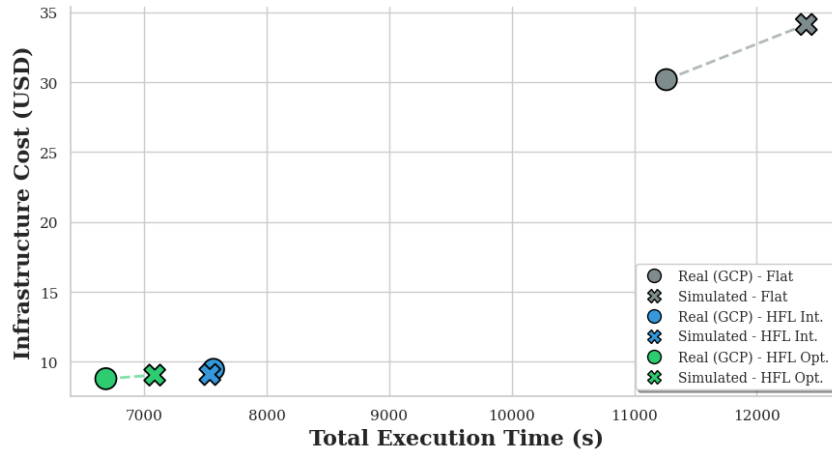


Figure 6.7: Validation of the simulator against the empirical deployment on Google Cloud Platform (GCP), for the 39-client So2Sat case study. Each configuration, namely the flat baseline and the hierarchical (HFL) architecture under the intuitive (Int.) and the optimized (Opt.) placements of the global parameter server, is plotted twice in the plane of total execution time against infrastructure cost: as a circle for the value measured on the real infrastructure and as a cross for the value produced by the simulator. The dashed segment joining each pair therefore expresses the deviation between the two, and the shorter it is, the closer the simulator reproduces the observed behaviour.

As illustrated in Figure 6.7, the simulated data for the flat, HFL Intuitive, and HFL Optimized configurations align closely with the empirical measurements, demonstrating fidelity for both total execution time and infrastructure cost. The variations observed are

attributed to fluctuations in network latency and transient FaaS cold starts inherent to the physical environment. Furthermore, these deviations reflect minor synchronization delays for message exchange and orchestration overheads that are not explicitly captured by the simulator.

6.2.4 Large-Scale Experiment

Recognizing the scalability limits of the flat architecture previously established, this experiment evaluates the efficiency gains within hierarchical strategies when scaling to 2,048 clients. We utilized the same proportional distribution of clients across four continents adopted in the empirical experiment.

In this large-scale context, we focus our analysis on the intercontinental communication between the global server and the edge servers. This choice is justified by the fact that our topology optimization primarily targets the strategic positioning of the global tier relative to the edge servers. While the client-to-edge links remain stable and unaffected by global placement, as they are confined to intra-region communication, the selection of the global hosting site directly dictates the latency and egress costs of the long-haul synchronization. Furthermore, to evaluate the long-term sustainability of the orchestration, the simulation configured the flat baseline to perform 56 global rounds, while the hierarchical models achieve an equivalent training effort using 28 global rounds (with $\kappa_2 = 2$ regional steps).

As illustrated in Figure 6.8, the transition from an always-on HFL model to the SHFL (Int.) version demonstrates a total cost reduction of approximately 16% (from \$37.82 to \$31.56). Since the distinction between these setups is strictly the billing model (always-on versus serverless), both maintain identical execution times, with efficiency driven primarily by the elimination of billing during idle periods for the aggregation functions. Both configurations still share an orchestration cost of \$1.76, as they both rely on a persistent instance to manage the training process throughout the entire execution.

In contrast, the SHFL Workflows (Opt.) configuration achieves superior efficiency by targeting communication bottlenecks and orchestration overheads. By leveraging the algorithmic selection of the optimized hosting region for the global parameter server, this setup reduces the intercontinental communication bottleneck by 29.7%, dropping from 3,026 s to 2,127 s. Furthermore, the transition to a native cloud workflow orchestration reduces the coordination cost to \$1.18. Consequently, the total infrastructure cost is reduced to \$30.92.

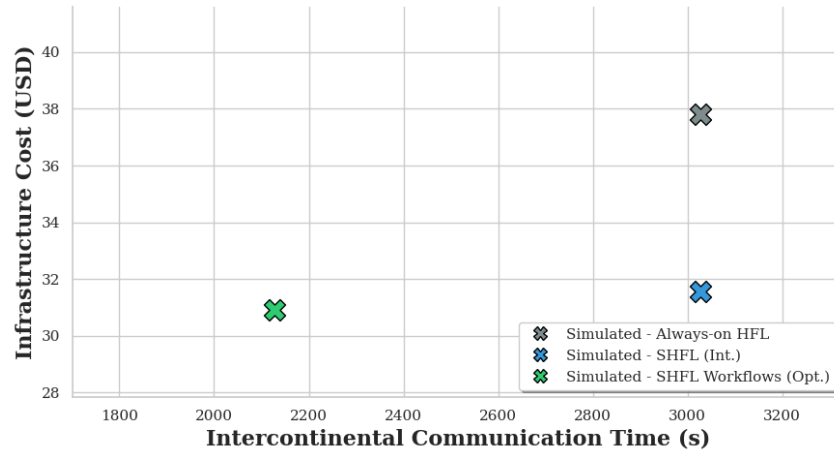


Figure 6.8: Comparison of Hierarchical Federated Learning (HFL) strategies at scale, showing intercontinental communication time against total infrastructure cost, for 2,048 clients over 56 rounds of equivalent training effort. The always-on HFL configuration relies on persistent instances; Serverless HFL (SHFL) Int. replaces them with functions while retaining a persistent orchestrator and the intuitive global placement; and SHFL Workflows Opt. adds the workflow-based orchestration and the optimized global placement.

These results suggest that for large-scale global FL, the combination of a hierarchical topology with serverless orchestration and region-aware placement offers a viable approach to managing transcontinental latency. While the financial gains at this scale are more incremental, the reduction in communication time, coupled with the elimination of the management burden of persistent infrastructure, indicates that this architecture can contribute to maintaining operational performance as the number of clients expands across regions.

Chapter 7

Conclusion and Future Work

FL enables collaborative model training without centralizing private data, but scaling it to a global federation exposes a bottleneck that the paradigm itself does not resolve: clients dispersed across continents must repeatedly synchronize with a parameter server, and the transcontinental links that carry this traffic are both slow and expensive. HFL mitigates this by interposing regional aggregation tiers, yet its conventional deployment on persistent virtual machines reintroduces a different inefficiency, since the intermittent nature of federated training leaves provisioned resources idle for long stretches while their cost accrues regardless. This work set out to address these two problems jointly, asking how a hierarchical federation should be deployed on commercial cloud infrastructure so that neither idle capacity nor long-haul communication dominates its cost.

Our answer combines two contributions. The first is an architectural one, presented in Chapter 4: a fully serverless orchestration model in which the responsibilities traditionally concentrated in a persistent parameter server are decomposed into three distinct mechanisms. Coordination is delegated to managed cloud workflows, aggregation to stateless functions, and model state to object storage, so that no component of the system needs to remain alive between training rounds. Crucially, these workflows are nested and regionally deployed, which confines the high-frequency coordination to intra-regional links and mirrors, at the orchestration layer, the communication structure that HierFAVG imposes on the model updates. To the best of our knowledge, this is the first work to employ workflow engines as a fully serverless orchestration layer for FL.

The second contribution, developed in Chapter 5, addresses the one exchange that the hierarchy cannot localize: the synchronization between the edge servers and the global parameter server. We showed that the region hosting this server is a consequential design decision rather than a matter of convenience, since neither the latency nor the egress cost

of a route is uniform across the globe, and we proposed a scoring function that selects it by jointly weighing inter-continental latency, network egress fees, and regional computation pricing.

The empirical evaluation, conducted through real deployments on GCP and reported in Chapter 6, substantiates both contributions. On the CIFAR-10 benchmark, the hierarchical topology reduced the time to reach the target accuracy by 24.4% relative to a flat baseline, and cut infrastructure cost by 93.4%, a reduction driven almost entirely by a 98.4% drop in network egress fees as traffic was confined within regional boundaries. In the world-scale So2Sat case study, spanning 39 cities across four continents, the hierarchical architecture reduced execution time by nearly 33%, and the region selection algorithm delivered a further 11.6% improvement by identifying a hosting region that geographic intuition would not have chosen, for a total reduction of 40.6% over the flat baseline. Calibrated simulations then extended the analysis to 2,048 clients, where the combination of serverless orchestration and region-aware placement reduced intercontinental communication time by 29.7%. Throughout, the orchestration layer itself remained economically negligible, costing less than a persistent coordinator would have consumed.

These results also delimit the conditions under which the approach pays off. The hierarchical topology is not universally advantageous: because each regional aggregation round introduces cold starts and workflow transitions, a configuration with too few global rounds, or with excessive regional aggregation relative to the model size, can be slower than the flat baseline it replaces. The advantage emerges only once the accumulated communication savings surpass these overheads, and the balance between κ_1 and κ_2 is what governs where that threshold lies.

Several limitations bound the scope of these findings and, in doing so, indicate where the work should go next. The evaluation was conducted on a single commercial provider, and while the architecture relies only on managed services with equivalents across the major clouds, the placement method was exercised against one pricing structure and one network. Extending it to multi-cloud orchestration would allow the global server to be placed not merely in the best region of a provider, but in the best region across providers, whose egress rates, as shown in Chapter 2, are structured differently from one another. The federation we studied was also synchronous and reliable, with every client participating in every round; real deployments must contend with stragglers and disconnections, and incorporating asynchronous aggregation strategies would make the architecture robust to the client heterogeneity that a global federation inevitably exhibits. The configuration

of κ_1 and κ_2 was, moreover, explored empirically: as the evaluation showed, it can make the difference between a hierarchical deployment that outperforms the flat baseline and one that is slower than it, yet the values were selected by sweeping the space rather than predicted in advance. Devising a method that estimates the most efficient configuration *a priori*, from the model size, the number of clients, and the measured communication and cold start costs, would spare a practitioner this search and let the region-aware placement of Chapter 5 be complemented by an equally principled choice of the aggregation frequencies. Finally, our concern throughout has been with the infrastructure rather than with the learning algorithm, which we deliberately held fixed in order to isolate architectural effects. Layering algorithmic optimizations onto it, such as robust aggregation, adaptive client selection, or update compression, would address the statistical dimension of the problem that this work has left untouched, and would compound with rather than substitute for the infrastructural gains reported here.

In closing, this dissertation has argued that the efficiency of a globally distributed federation is determined not only by the algorithm that trains it but by the infrastructure on which it runs and the geography in which that infrastructure is placed. By aligning the orchestration of a hierarchical federation with the serverless execution model, and its topology with the pricing and latency structure of the underlying network, we have shown that both the operational burden and the financial cost of global-scale FL can be substantially reduced.

Bibliography

- [1] ABAD, M. S. H.; OZFATURA, E.; GUNDUZ, D.; ERCETIN, O. Hierarchical federated learning across heterogeneous cellular networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), pp. 8866–8870.
- [2] AMAZON WEB SERVICES. Amazon ebs documentation. <https://docs.aws.amazon.com/ebs/>, 2026. Accessed: 2026-06-22.
- [3] AMAZON WEB SERVICES. Amazon ec2 documentation. <https://docs.aws.amazon.com/ec2/>, 2026. Accessed: 2026-06-18.
- [4] AMAZON WEB SERVICES. Amazon s3 documentation. <https://docs.aws.amazon.com/s3/>, 2026. Accessed: 2026-06-15.
- [5] AMAZON WEB SERVICES. Amazon sns documentation. <https://docs.aws.amazon.com/sns/>, 2026. Accessed: 2026-06-15.
- [6] AMAZON WEB SERVICES. Amazon web services cloud platform. <https://aws.amazon.com/>, 2026. Accessed: 2026-06-15.
- [7] AMAZON WEB SERVICES. Aws lambda documentation. <https://docs.aws.amazon.com/lambda/>, 2026. Accessed: 2026-06-15.
- [8] AMAZON WEB SERVICES. Aws step functions documentation. <https://docs.aws.amazon.com/step-functions/>, 2026. Accessed: 2026-06-15.
- [9] ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. *Commun. ACM* 53, 4 (Apr. 2010), 50–58.
- [10] BRUM, R. C.; SENS, P.; ARANTES, L.; CASTRO, M. C.; DE A. DRUMMOND, L. M. Optimizing execution time and costs of cross-silo federated learning applications with datasets on different cloud providers. In *2022 IEEE 34th International*

- Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (2022), pp. 253–262.
- [11] BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25, 6 (2009), 599–616.
- [12] CASTRO, P.; ISHAKIAN, V.; MUTHUSAMY, V.; SLOMINSKI, A. The rise of serverless computing. *Commun. ACM* 62, 12 (Nov. 2019), 44–54.
- [13] CHADHA, M.; JENSEN, A.; GU, J.; ABBOUD, O.; GERNDT, M. Apodotiko: Enabling efficient serverless federated learning in heterogeneous environments. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (2024), pp. 206–215.
- [14] CHIANG, M.; ZHANG, T. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal* 3, 6 (2016), 854–864.
- [15] ELZOHAIY, M.; CHADHA, M.; JINDAL, A.; GRAFBERGER, A.; GU, J.; GERNDT, M.; ABBOUD, O. Fedlesscan: Mitigating stragglers in serverless federated learning. In *2022 IEEE International Conference on Big Data (Big Data)* (2022), pp. 1230–1237.
- [16] FILHO, C. P.; MARQUES, E.; CHANG, V.; DOS SANTOS, L.; BERNARDINI, F.; PIRES, P. F.; OCHI, L.; DELICATO, F. C. A systematic literature review on distributed machine learning in edge computing. *Sensors* 22, 7 (2022).
- [17] GAFF, B. M.; SUSSMAN, H. E.; GEETTER, J. Privacy and big data. *Computer* 47, 6 (2014), 7–9.
- [18] GAN, M.; LI, L.; ALAM, S.; LIU, L.; LIU, L.; ZHANG, M.; CAO, Z. Geoff: A framework for efficient geo-distributed cross-device federated learning. In *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications* (2025), pp. 1–10.
- [19] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.
- [20] GOOGLE. Cloud run functions documentation. <https://docs.cloud.google.com/functions/docs>, 2026. Accessed: 2026-06-15.
- [21] GOOGLE. Cloud run pricing. <https://cloud.google.com/run/pricing>, 2026. Accessed: 2026-06-15.

- [22] GOOGLE. Gcp app engine documentation. <https://docs.cloud.google.com/appengine/docs>, 2026. Accessed: 2026-06-18.
- [23] GOOGLE. Gcp cloud storage documentation. <https://docs.cloud.google.com/storage/docs/>, 2026. Accessed: 2026-06-15.
- [24] GOOGLE. Gcp pub/sub documentation. <https://docs.cloud.google.com/pubsub/docs?>, 2026. Accessed: 2026-06-15.
- [25] GOOGLE. Gcp workflows documentation. <https://cloud.google.com/workflows>, 2026. Accessed: 2026-06-15.
- [26] GOOGLE. Google cloud platform. <https://cloud.google.com>, 2026. Accessed: 2026-06-15.
- [27] GOOGLE. Vpc network pricing. <https://cloud.google.com/vpc/network-pricing>, 2026. Accessed: 2026-06-15.
- [28] GRAFBERGER, A.; CHADHA, M.; JINDAL, A.; GU, J.; GERNDT, M. Fedless: Secure and scalable federated learning using serverless computing. In *2021 IEEE International Conference on Big Data (Big Data)* (2021), pp. 164–173.
- [29] HARD, A.; RAO, K.; MATHEWS, R.; RAMASWAMY, S.; BEAUFAYS, F.; AUGENSTEIN, S.; EICHNER, H.; KIDDON, C.; RAMAGE, D. Federated learning for mobile keyboard prediction, 2019.
- [30] HASHICORP. Terraform documentation. <https://developer.hashicorp.com/terraform/docs>, 2026. Accessed: 2026-06-15.
- [31] HELLERSTEIN, J. M.; FALEIRO, J. M.; GONZALEZ, J.; SCHLEIER-SMITH, J.; SREEKANTI, V.; TUMANOV, A.; WU, C. Serverless computing: One step forward, two steps back. *ArXiv abs/1812.03651* (2018).
- [32] HSU, T.-M. H.; QI, H.; BROWN, M. Measuring the effects of non-identical data distribution for federated visual classification, 2019.
- [33] HUDSON, N.; HAYOT-SASSON, V.; BABUJI, Y.; BAUGHMAN, M.; PAULOSKI, J. G.; CHARD, R.; FOSTER, I.; CHARD, K. Flight: A faas-based framework for complex and hierarchical federated learning. *Future Generation Computer Systems* 174 (2026), 107998.

- [34] JAIN, P.; KUMAR, S.; WOODERS, S.; PATIL, S. G.; GONZALEZ, J. E.; STOICA, I. Skyplane: Optimizing transfer cost and throughput using {Cloud-Aware} overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1375–1389.
- [35] JIANG, J.; GAN, S.; LIU, Y.; WANG, F.; ALONSO, G.; KLIMOVIC, A.; SINGLA, A.; WU, W.; ZHANG, C. Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data* (New York, NY, USA, 2021), SIGMOD '21, Association for Computing Machinery, p. 857–871.
- [36] JIANG, J.; GAN, S.; LIU, Y.; WANG, F.; ALONSO, G.; KLIMOVIC, A.; SINGLA, A.; WU, W.; ZHANG, C. Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data* (New York, NY, USA, 2021), SIGMOD '21, Association for Computing Machinery, p. 857–871.
- [37] JIANG, Z.; WANG, W.; LI, B.; LI, B. Pisces: efficient federated learning via guided asynchronous training. SoCC '22, Association for Computing Machinery, p. 370–385.
- [38] JOHNSTON, P. A simple definition of serverless. <https://pauldjohnston.medium.com/a-simple-definition-of-serverless-8492adfb175a>, 2017. Accessed: 2026-06-22.
- [39] KOTSEHUB, N.; BAUGHMAN, M.; CHARD, R.; HUDSON, N.; PATROS, P.; RANA, O.; FOSTER, I.; CHARD, K. Flox: Federated learning with faas at the edge. In *2022 IEEE 18th International Conference on e-Science (e-Science)* (2022), pp. 11–20.
- [40] KRIZHEVSKY, A. Learning multiple layers of features from tiny images. Tech. rep., University of Toronto, 2009. Technical Report.
- [41] LECUN, Y.; CORTES, C.; BURGESS, C. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [42] LI, Q.; WEN, Z.; WU, Z.; HU, S.; WANG, N.; LI, Y.; LIU, X.; HE, B. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2023), 3347–3366.
- [43] LI, Z.; CHARD, R.; BABUJI, Y.; GALEWSKY, B.; SKLUZACEK, T. J.; NAGAITSEV, K.; WOODARD, A.; BLAISZIK, B.; BRYAN, J.; KATZ, D. S.; FOSTER, I.;

- CHARD, K. funcx: Federated function as a service for science. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4948–4963.
- [44] LIU, L.; ZHANG, J.; SONG, S.; LETAIEF, K. B. Client-edge-cloud hierarchical federated learning. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)* (2020), pp. 1–6.
- [45] LIU, L.; ZHANG, J.; SONG, S.; LETAIEF, K. B. Hierarchical federated learning with quantization: Convergence analysis and system design. *IEEE Transactions on Wireless Communications* 22, 1 (2023), 2–18.
- [46] LIU, Y.; QU, Z.; WANG, J. Compressed hierarchical federated learning for edge-level imbalanced wireless networks. *IEEE Transactions on Computational Social Systems* 12, 5 (2025), 3131–3142.
- [47] LUO, S.; CHEN, X.; WU, Q.; ZHOU, Z.; YU, S. Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. *IEEE Transactions on Wireless Communications* 19, 10 (2020), 6535–6548.
- [48] MCMAHAN, B.; MOORE, E.; RAMAGE, D.; HAMPSON, S.; ARCAS, B. A. Y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (20–22 Apr 2017), A. Singh and J. Zhu, Eds., vol. 54 of *Proceedings of Machine Learning Research*, PMLR, pp. 1273–1282.
- [49] MELL, P.; GRANCE, T. *The NIST definition of cloud computing*. 2011.
- [50] MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux J.* 2014, 239 (Mar. 2014).
- [51] MICROSOFT. Azure blob storage documentation. <https://learn.microsoft.com/en-us/azure/storage/blobs/>, 2026. Accessed: 2026-06-15.
- [52] MICROSOFT. Azure functions documentation. <https://learn.microsoft.com/en-us/azure/azure-functions/>, 2026. Accessed: 2026-06-15.
- [53] MICROSOFT. Azure logic apps documentation. <https://learn.microsoft.com/en-us/azure/logic-apps/>, 2026. Accessed: 2026-06-15.
- [54] MICROSOFT. Azure service bus documentation. <https://learn.microsoft.com/en-us/azure/service-bus-messaging/>, 2026. Accessed: 2026-06-15.

- [55] MICROSOFT. Bandwidth pricing. <https://azure.microsoft.com/en-us/pricing/details/bandwidth/>, 2026. Accessed: 2026-07-13.
- [56] MICROSOFT. Microsoft azure cloud computing platform. <https://azure.microsoft.com/>, 2026. Accessed: 2026-06-15.
- [57] MITCHELL, T. M. *Machine learning*, vol. 1. McGraw-hill New York, 1997.
- [58] MORRIS, K. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, Inc., 2016.
- [59] NG, J. S.; LIM, W. Y. B.; XIONG, Z.; CAO, X.; JIN, J.; NIYATO, D.; LEUNG, C.; MIAO, C. Reputation-aware hedonic coalition formation for efficient serverless hierarchical federated learning. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2022), 2675–2686.
- [60] PADHYE, J.; FIROIU, V.; TOWSLEY, D.; KUROSE, J. Modeling tcp throughput: a simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.* 28, 4 (Oct. 1998), 303–314.
- [61] PECK PINHEIRO, P.; BATISTA BATTAGLINI, H. Artificial intelligence and data protection: A comparative analysis of ai regulation through the lens of data protection in the eu and brazil. *GRUR International* 71, 10 (10 2022), 924–932.
- [62] POWLES, J.; HODSON, H. Google deepmind and healthcare in an age of algorithms. *Health and Technology* 7, 4 (Dec 2017), 351–367.
- [63] RAUSCH, T.; PHAM, S. N. S.; SCHMIDT, P. L. S.; DUSTDAR, S. Enabling HPC scientific workflows for serverless. In *2020 IEEE International Conference on Cloud Computing (CLOUD)* (2020), pp. 406–410.
- [64] RYDNING, D. R.-J. G.-J.; REINSEL, J.; GANTZ, J. The digitization of the world from edge to core. *Framingham: International Data Corporation* 16 (2018), 1–28.
- [65] WANG, J.; WANG, S.; CHEN, R.-R.; JI, M. Demystifying why local aggregation helps: Convergence analysis of hierarchical sgd. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 8 (Jun. 2022), 8548–8556.
- [66] WANG, L.; LI, M.; ZHANG, Y.; RISTENPART, T.; SWIFT, M. Peeking behind the curtains of serverless platforms. In *2018 USENIX annual technical conference (USENIX ATC 18)* (2018), pp. 133–146.

-
- [67] YANG, Q.; LIU, Y.; CHEN, T.; TONG, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (Jan. 2019).
- [68] YOUSEFF, L.; BUTRICO, M.; DA SILVA, D. Toward a unified ontology of cloud computing. In *2008 Grid Computing Environments Workshop* (2008), pp. 1–10.
- [69] ZHU, X.; HU, J.; QIU, C.; SHI, Y.; BAGHERI, H.; KANG, J.; LI, H.; MOU, L.; ZHANG, G.; HÄBERLE, M.; HAN, S.; HUA, Y.; HUANG, R.; HUGHES, L.; SUN, Y.; SCHMITT, M.; WANG, Y. New: So2sat lcz42, 2019.

APPENDIX A – Complete Experimental Data

This appendix provides the raw data underlying the evaluation of the proposed architecture, organized into two groups. The first comprises the individual execution runs collected during the empirical validation on GCP, which serve as the foundation for the averaged results and visualizations discussed in Chapter 6. Table A.1 outlines the per-round wall-clock times used to evaluate communication efficiency (Section 6.1.2); Table A.2 presents the end-to-end execution times required to reach the target accuracy for MNIST and CIFAR-10 (Section 6.1.3); and Table A.3 details the performance of the world-scale So2Sat case study, comparing the intuitive and the optimized global server placement (Section 6.1.4).

The second group comprises the measurements that instantiate the three metrics of the region selection algorithm of Chapter 5. Table A.4 reports the inter-region communication latency obtained through the metering function described in Section 5.1.2, Table A.6 reports the egress rates collected from the GCP network pricing tables, and Table A.5 reports the regional computation costs collected from the Cloud Run pricing tables. Together, these three tables constitute the complete input to the scoring function of Equation 5.1, and allow the selection of the optimized region to be reproduced independently.

Table A.1: Complete empirical results for the communication efficiency evaluation (corresponding to Figure 6.2), comparing the wall-clock time of the flat and the hierarchical architectures across different numbers of training rounds.

Dataset	Architecture	Local Steps	Rounds (k, κ_2)	Time (s)	Improvement (s)
MNIST	Flat	6	1	11.374	–
MNIST	Flat	6	1	10.405	–
MNIST	Flat	6	1	11.219	–

continued on next page

Table A.1 – continued from previous page

Dataset	Architecture	Local Steps	Rounds (k, κ_2)	Time (s)	Improvement (s)
MNIST	Flat	6	5	52.864	–
MNIST	Flat	6	5	53.323	–
MNIST	Flat	6	5	50.458	–
MNIST	Flat	6	10	111.000	–
MNIST	Flat	6	10	106.000	–
MNIST	Flat	6	10	106.000	–
MNIST	Hierarchical	6	1	47.161	-35.942
MNIST	Hierarchical	6	1	35.737	-24.518
MNIST	Hierarchical	6	1	25.297	-14.078
MNIST	Hierarchical	6	5	47.165	3.293
MNIST	Hierarchical	6	5	47.272	3.186
MNIST	Hierarchical	6	5	45.893	4.565
MNIST	Hierarchical	6	10	77.000	29.000
MNIST	Hierarchical	6	10	87.000	19.000
MNIST	Hierarchical	6	10	77.000	29.000
CIFAR-10	Flat	6	1	14.968	–
CIFAR-10	Flat	6	1	13.954	–
CIFAR-10	Flat	6	1	12.325	–
CIFAR-10	Flat	6	5	59.769	–
CIFAR-10	Flat	6	5	57.205	–
CIFAR-10	Flat	6	5	57.341	–
CIFAR-10	Flat	6	10	116.000	–
CIFAR-10	Flat	6	10	128.000	–
CIFAR-10	Flat	6	10	114.000	–
CIFAR-10	Hierarchical	6	1	29.764	-17.439
CIFAR-10	Hierarchical	6	1	27.596	-15.271

continued on next page

Table A.1 – continued from previous page

Dataset	Architecture	Local Steps	Rounds (k, κ_2)	Time (s)	Improvement (s)
CIFAR-10	Hierarchical	6	1	27.254	-14.929
CIFAR-10	Hierarchical	6	5	58.556	-1.351
CIFAR-10	Hierarchical	6	5	58.151	-0.946
CIFAR-10	Hierarchical	6	5	59.584	-2.379
CIFAR-10	Hierarchical	6	10	99.000	15.000
CIFAR-10	Hierarchical	6	10	111.000	3.000
CIFAR-10	Hierarchical	6	10	110.000	4.000

Table A.2: Complete empirical results for the time-to-accuracy evaluation (corresponding to Figure 6.3), comparing the flat and the hierarchical architectures for MNIST (85% target) and CIFAR-10 (70% target).

Dataset	Architecture	Local Steps (κ_1)	Reg. Rounds (κ_2)	Global Rounds (k)	Time (s)
MNIST	Flat	60	-	7	443.7
MNIST	Flat	60	-	10	623.2
MNIST	Flat	60	-	6	382.4
MNIST	Hierarchical	30	2	6	305.9
MNIST	Hierarchical	30	2	6	346.1
MNIST	Hierarchical	30	2	6	356.4
MNIST	Hierarchical	20	3	5	390.1
MNIST	Hierarchical	20	3	5	309.9
MNIST	Hierarchical	20	3	5	302.4
MNIST	Hierarchical	15	4	4	430.1
MNIST	Hierarchical	15	4	4	392.5
MNIST	Hierarchical	15	4	4	377.0
MNIST	Hierarchical	10	6	3	474.3
MNIST	Hierarchical	10	6	3	424.8
MNIST	Hierarchical	10	6	3	402.2
CIFAR-10	Flat	50	-	23	5030.7
CIFAR-10	Flat	50	-	23	4988.5
CIFAR-10	Flat	50	-	23	4997.4
CIFAR-10	Hierarchical	25	2	18	3622.5
CIFAR-10	Hierarchical	25	2	18	4006.6
CIFAR-10	Hierarchical	25	2	18	3716.1
CIFAR-10	Hierarchical	10	5	22	8700.8
CIFAR-10	Hierarchical	10	5	22	9367.1
CIFAR-10	Hierarchical	10	5	22	10858.7

Table A.3: Complete empirical results for the time-to-accuracy evaluation of the So2Sat global case study (corresponding to Figure 6.5), comparing the intuitive and the optimized placements of the global parameter server (50% target accuracy).

Dataset	Architecture	Location	Local Steps (κ_1)	Reg. Rounds (κ_2)	Global Rounds (k)	Time (s)
So2Sat	Hierarchical (Int.)	europe-north2	25	2	14	6859.0
So2Sat	Hierarchical (Int.)	europe-north2	25	2	14	8282.0
So2Sat	Hierarchical (Int.)	europe-north2	25	2	14	7556.0
So2Sat	Hierarchical (Opt.)	us-central1	25	2	14	6420.0
So2Sat	Hierarchical (Opt.)	us-central1	25	2	14	7017.0
So2Sat	Hierarchical (Opt.)	us-central1	25	2	14	6623.0

Table A.4: Measured inter-region communication latency (in milliseconds) used as the metric $\mathcal{L}$ of the region selection algorithm. Each value is the average of five executions of the metering function deployed in the source region, downloading a test payload from a storage bucket in the destination region. Rows denote the source region and columns the destination.

Source	Asia	Australia	Europe	S. America	US
Asia	456	705	1247	1408	708
Australia	728	425	1074	1490	708
Europe	1466	1469	229	1038	541
S. America	1474	1513	817	337	542
US	868	956	602	673	264

Table A.5: Regional computation cost used as the metric $\mathcal{P}$ of the region selection algorithm, collected from the Google Cloud Run pricing tables. Values are expressed in US dollars per second of execution under the request-based billing model, the one adopted in this work.

Region	Cost	Region	Cost	Region	Cost
us-central1	0.000184	europa-north1	0.000184	asia-east1	0.000184
us-east1	0.000184	europa-north2	0.000184	asia-east2	0.000258
us-east4	0.000184	europa-southwest1	0.000184	asia-northeast1	0.000184
us-east5	0.000184	europa-west1	0.000184	asia-northeast2	0.000184
us-south1	0.000184	europa-west2	0.000258	asia-northeast3	0.000258
us-west1	0.000184	europa-west3	0.000258	asia-southeast1	0.000258
us-west2	0.000258	europa-west4	0.000184	asia-southeast2	0.000258
us-west3	0.000258	europa-west6	0.000258	asia-south1	0.000184
us-west4	0.000258	europa-west8	0.000184	asia-south2	0.000258
northamerica-northeast1	0.000258	europa-west9	0.000184	australia-southeast1	0.000258
northamerica-northeast2	0.000258	europa-west10	0.000258	australia-southeast2	0.000258
northamerica-south1	0.000184	europa-west12	0.000258	me-central1	0.000258
southamerica-east1	0.000258	europa-central2	0.000258	me-central2	0.000258
southamerica-west1	0.000258			me-west1	0.000184
				africa-south1	0.000258

Table A.6: Inter-region egress rates used as the metric $\mathcal{E}$ of the region selection algorithm, in US dollars per GiB. Because Google Cloud Platform (GCP) prices each transfer by the pair of continents involved, the complete region-to-region matrix collapses to the values below. Rows denote the source continent and columns the destination.

Source	N. Am.	S. Am.	Europe	Asia	Australia	Mid. East	Africa
North America	0.02	0.14	0.05	0.08	0.10	0.11	0.11
South America	0.14	0.14	0.14	0.14	0.14	0.14	0.14
Europe	0.05	0.14	0.02	0.08	0.10	0.11	0.11
Asia	0.08	0.14	0.08	0.08	0.10	0.11	0.11
Australia	0.10	0.14	0.10	0.10	0.08	0.11	0.11
Middle East	0.11	0.14	0.11	0.11	0.11	0.11	0.11
Africa	0.11	0.14	0.11	0.11	0.11	0.11	0.11

APPENDIX B - Publications

This appendix lists the publications produced during the course of this master's program. The first is the paper from which this dissertation directly derives; the remaining ones stem from collaborations carried out in parallel.

Derived from this dissertation

Pereira, M. M.; Drummond, L. M. A. Optimizing Global Federated Learning: A Serverless Hierarchical Approach with Region-Aware Placement. In *Proceedings of the 16th International Conference on Cloud Computing and Services Science (CLOSER)*, pp. 108–119. SciTePress, 2026. *Nominated for the Best Paper Award*. DOI: 10.5220/0015043000004039.

Other publications

Santos, B. G. V.; Pereira, M. M.; Drummond, L. M. A.; Viterbo Filho, J. Análise de um Log Real de Jobs visando a Previsão de Tempos de Espera para Execução em um Cluster de Alto Desempenho. In *Anais da X Escola Regional de Alto Desempenho da Região Sudeste (ERAD-SE)*, Niterói, 2025. DOI: 10.5753/eradse.2025.16956.

Pereira, M. M.; Santos, B. G. V.; Drummond, L. M. A.; Gadelha, A. InPEX 2026: Ciência Global e Computação de Alto Desempenho em Niterói. *SBC Horizontes*, May 2026.

Under review

Nunes, A. L.; Gallo, B.; Pereira, M. M.; Viterbo, J.; Portella, F. A.; Estrela, P. J. B.; Malini, R. Q.; Drummond, L. M. A. Wise-Burst: a Decision Support Tool for Cloud Bursting. *IEEE Transactions on Cloud Computing* (under major revision).