

UNIVERSIDADE FEDERAL FLUMINENSE

ELBE ALVES MIRANDA

**O SPARQL Pode Falar em Português?
Respondendo a Perguntas em Linguagem Natural
Direcionadas a Grafos de Conhecimento**

NITERÓI

2023

ELBE ALVES MIRANDA

**O SPARQL Pode Falar em Português?
Respondendo a Perguntas em Linguagem Natural
Direcionadas a Grafos de Conhecimento**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação.

Orientadora:

ALINE MARINS PAES CARVALHO

Coorientador:

DANIEL CARDOSO MORAES DE OLIVEIRA

NITERÓI

2023

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

M672s Miranda, Elbe Alves
 O SPARQL Pode Falar em Português? Respondendo a Perguntas
 em Linguagem Natural Direcionadas a Grafos de Conhecimento /
 Elbe Alves Miranda. - 2023.
 81 f.: il.

 Orientador: Aline Marins Paes Carvalho.
 Coorientador: Daniel Cardoso Moraes De Oliveira.
 Dissertação (mestrado)-Universidade Federal Fluminense,
 Instituto de Computação, Niterói, 2023.

 1. Grafo. 2. Ferramenta de busca. 3. Produção intelectual.
 I. Carvalho, Aline Marins Paes, orientadora. II. De Oliveira,
 Daniel Cardoso Moraes, coorientador. III. Universidade Federal
 Fluminense. Instituto de Computação. IV. Título.

CDD - XXX

ELBE ALVES MIRANDA

O SPARQL Pode Falar em Português? Respondendo a Perguntas em Linguagem
Natural Direcionadas a Grafos de Conhecimento

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação.

Aprovada em dezembro de 2023.

BANCA EXAMINADORA



Documento assinado digitalmente
ALINE MARINS PAES CARVALHO
Data: 08/12/2023 16:22:18-0300
Verifique em <https://validar.iti.gov.br>

Prof^a. Aline Marins Paes Carvalho - Orientadora, UFF



Documento assinado digitalmente
DANIEL CARDOSO MORAES DE OLIVEIRA
Data: 12/12/2023 15:46:45-0300
Verifique em <https://validar.iti.gov.br>

Prof. Daniel Cardoso Moraes de Oliveira - Coorientador, UFF



Documento assinado digitalmente
MARCOS VINICIUS NAVES BEDO
Data: 12/12/2023 10:27:34-0300
Verifique em <https://validar.iti.gov.br>

Prof. Marcos Vinícius Naves Bêdo, UFF



Documento assinado digitalmente
RONALDO DOS SANTOS MELLO
Data: 08/12/2023 17:54:07-0300
CPF: ***.693.660-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Ronaldo dos Santos Mello, UFSC

Niterói
2023

Dedico este trabalho aos meus familiares, especialmente ao meu pai Franciso e à minha mãe Alzirene, que mesmo diante de muitas dificuldades nunca negligenciaram a educação dos filhos. Amo vocês!

Agradecimentos

Agradeço à minha orientadora Aline Paes e também ao meu co-orientador Daniel de Oliveira por me guiarem nesta jornada desafiadora. Suas orientações e conselhos foram de grande ajuda. Agradeço especialmente à compreensão que tiveram nos momentos mais difíceis. Isso só justifica o grande caráter humano que vocês dois têm, além dos grandes profissionais que são!

Um agradecimento especial à minha amada esposa Lidianne Miranda, que sempre esteve ao meu lado, tanto nos momentos mais felizes quanto nos momentos mais difíceis, sem você tudo seria muito mais difícil e este trabalho teria sido impossível.

Não tenho como deixar de agradecer às minhas lindas filhas Júlia e Louise, por serem tão compreensíveis durante as minhas ausências e por tornarem os meus dias mais felizes!

Agradeço também aos meus pais e às minhas queridas irmãs Tatiane, Taylane e Tayne. Vocês são exemplos de perseverança e pensar em vocês me ajudou a não desistir.

Por último e não menos importante, agradeço ao meu Deus e Criador, Jeová, pela vida e pela oportunidade defender e levar o Seu nome!

Resumo

A busca de estratégias para respostas a perguntas em linguagem natural a partir de um Grafo de Conhecimento (KG) é um campo conhecido como Question Answering (QA) em Grafos de Conhecimento (KGQA). Tal tarefa permite que os usuários obtenham respostas sem precisar ter conhecimento em uma linguagem de consulta de KG específica, como o SPARQL. Embora a maioria das soluções existentes se concentre em treinar modelos de Aprendizado de Máquina (ML) para converter perguntas em inglês para consultas em SPARQL, poucas iniciativas foram feitas para idiomas além do inglês, como o português, que é o sexto idioma mais falado no mundo e apresenta seus próprios desafios linguísticos. Infelizmente, essas limitações incluem um pequeno número de conjuntos de dados para reproduzir as soluções de ML baseadas em inglês em outros idiomas. Em vez de treinar uma solução completa de ML de ponta a ponta, este trabalho apresenta uma abordagem modular para a tarefa de KGQA em português, que se baseia em cinco componentes: (i) Analisador Sintático, (ii) Classificação do Tipo de Pergunta, (iii) Mapeamento de Conceitos, (iv) Geração de Consultas e (v) Ranqueamento de Consultas. Nossas contribuições incluem modelos treinados para classificação de perguntas e classificação de consultas, especificamente adaptados para o idioma português, oferecendo uma solução abrangente para responder perguntas em linguagem natural a partir de KGs, além de um novo modelo de Ligação de Relações para o português, o PTRL. Em experimentos conduzidos usando os conjuntos de dados QALD e LCQuAD, a solução proposta alcançou um F1-score geral de 41.9% no QALD e de 45.8% no LCQuAD, superando uma pontuação de referência de 28.5%. Até onde temos conhecimento, essa é a primeira solução de KGQA projetada para o português que utiliza os conjuntos de dados padrão QALD e LCQuAD.

Palavras-chave: Perguntas e Respostas, Grafo de Conhecimento, Português, SPARQL, QALD, LCQuAD

Abstract

Answering natural language questions from a Knowledge Graph (KG) is a field known as Knowledge Graph Question Answering (KGQA), which enables users to pose their queries without any expertise in a specific KG query language, such as SPARQL. While the majority of existing solutions focus on training Machine Learning (ML) models that convert English questions to SPARQL queries, only a bunch of initiatives have been designed for languages other than English (such as Portuguese). As an additional challenge, there is also a small number of datasets to reproduce ML solutions based on English in other languages. Therefore, instead of training a complete end-to-end solution, this study presents a modular approach for the task of KGQA regarding Portuguese questions. It is based on five components: (i) Syntax Analyzer, (ii) Question Type Classification, (iii) Concept Mapping, (iv) Query Generation, and (v) Query Ranking. Our contributions include trained models for question classification and query ranking that are tailored explicitly for the Portuguese language, offering a comprehensive solution for answering questions in natural language from KGs. We also proposed and validated a new Relation Linking model for Portuguese, called PTRL. In experiments conducted using the QALD and LCQuAD datasets, our solution achieved an average F1-score of 41.9% on QALD and 45.8% on LCQuAD, outperforming the baseline score of 28.5%. To our knowledge, this is the first KGQA solution designed for Portuguese that sets a benchmark on QALD and LCQuAD datasets.

Keywords: Question Answering, Knowledge Graph, Portuguese, SPARQL, QALD, LCQuAD

Lista de Figuras

1	Exemplo de um subgrafo do Grafo de Conhecimento DBpedia.	9
2	Grafo representando o fato simples: “ <i>Ayrton Senna nasceu em São Paulo</i> ”.	9
3	Grafo representando o fato complexo: “ <i>Ayrton Senna foi o vencedor do GP de Mônaco, ocorrido em 1993</i> ”	10
4	Página do Ayrton Senna na Wikipedia.	11
5	Página do Ayrton Senna na DBpedia.	12
6	A resposta à pergunta “ <i>Onde nasceu o piloto Ayrton Senna?</i> ” está a um salto de distância da entidade “ <i>dbr:Ayrton_Senna</i> ” identificada na pergunta.	13
7	A resposta à pergunta: “ <i>Quem foi o diretor do documentário de Ayrton Senna?</i> ” está a dois saltos de distância do nó “ <i>dbr:Ayrton_Senna</i> ” identificado na pergunta.	14
8	Diagrama ilustrativo de uma <i>Long Short Term Memory</i> (LSTM).(OLAH, s.d.)	16
9	Sistema baseado em componentes para a tarefa de <i>Knowledge Graph Question Answering</i> (KGQA) em português.	27
10	Características sintáticas referentes à frase: <i>Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?</i>	28
11	Árvore de Dependência Sintática referente à pergunta: “ <i>Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?</i> ”	30
12	Esquema de funcionamento do método de Ligação de Relações RNLIWOD adaptado para o português	36
13	Resumo do funcionamento do ptRL para Ligação de Relações.	37

-
- 14 Comparação entre a arquitetura de uma LSTM, acima, e uma *Tree Long Short Term Memory* (Tree-LSTM), abaixo. (Reproduzido de (TAI; SOCHER; MANNING, 2015)) 43
 - 15 Representação em árvore de uma tripla de uma consulta SPARQL que responde à pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*” 44
 - 16 Processo de cálculo da similaridade entre uma pergunta e a sua respectiva consulta em *SPARQL Protocol and RDF Query Language* (SPARQL). . . . 44
 - 17 Comparativo do desempenho do *Portuguese Knowledge Graph Question Answering* (KGQA_{PT}) com e sem o uso do modelo de Ligação de Relações *Portuguese Relation Linking* (PTRL) 56

Lista de Tabelas

1	Classificador do Tipo de Pergunta com <i>Term Frequency/Inverse Document Frequency</i> (TF/IDF) para o <i>dataset Largescale Complex Question Answering Dataset</i> (LCQuAD)	48
2	Classificador do Tipo de Pergunta com TF/IDF para o <i>dataset Question Answering over Linked Data</i> (QALD)	48
3	Classificador do Tipo de Pergunta com <i>embedding</i> para o <i>dataset LCQuAD</i>	49
4	Classificador do Tipo de Pergunta com <i>embedding</i> para o <i>dataset QALD</i>	49
5	Classificador de Relação Dupla com TF/IDF para o <i>dataset LCQuAD</i> .	50
6	Classificador de Relação Dupla com <i>embedding</i> para o <i>dataset LCQuAD</i>	51
7	Hiper-parâmetros para o modelo Tree-LSTM	51
8	Avaliação do QASPARQL no <i>dataset QALD-7</i>	52
9	Avaliação do KGQA _{PT} no <i>dataset QALD-7</i> sem PTRL	53
10	Avaliação do KGQA _{PT} no conjunto de teste do <i>dataset LCQuAD</i> sem PTRL	53
11	Avaliação do KGQA _{PT} em todo o <i>dataset LCQuAD</i> sem PTRL	54
12	Avaliação do KGQA _{PT} no <i>dataset QALD-7</i> com PTRL	54
13	Avaliação do KGQA _{PT} no conjunto de teste do <i>dataset LCQuAD</i> com PTRL	55
14	Avaliação do KGQA _{PT} em todo o <i>dataset LCQuAD</i> com PTRL	55

Lista de Abreviaturas e Siglas

KGQA_{PT} *Portuguese Knowledge Graph Question Answering*

PTRL *Portuguese Relation Linking*

AM *Aprendizado de Máquina*

API *Application Programming Interface*

BCLs *Bases de Conhecimento Lexicais*

CC *Construção da Consulta*

CLEF *Conference and Labs of the Evaluation Forum*

DL *Deep Learning*

EL *Entity Linking*

FN *False Negatives*

FP *False Positives*

GC *Grafo de Conhecimento*

GCs *Grafos de Conhecimento*

GPT *Generative Pretrained Transformer*

IR *Information Retrieval*

IRI *Internationalized Resource Identifier*

KG *Knowledge Graph*

KGQA *Knowledge Graph Question Answering*

KGs *Knowledge Graphs*

LCQuAD *Largescale Complex Question Answering Dataset*

LE *Ligação de Entidade*

LLMs *Large Language Models*

LR *Ligação de Relação*

LSTM *Long Short Term Memory*

NLP *Natural Language Processing*

NMT *Neural Machine Translation*

NN *Neural Network*

PLN *Processamento de Linguagem Natural*

POS-Tagging *Part-Of-Speech Tagging*

QA *Question Answering*

QALD *Question Answering over Linked Data*

QB *Query Building*

QR *Query Ranking*

RC *Rankeamento de Consulta*

RDF *Resource Description Framework*

RF *Random Forest*

RL *Relation Linking*

RN *Rede Neural*

RNN *Recurrent Neural Network*

RNNs *Recurrent Neural Networks*

SP *Semantic Parser*

SPARQL *SPARQL Protocol and RDF Query Language*

SQG *SPARQL Query Generator*

SQuAD *Stanford Question Answering Dataset*

SVM *Support Vector Machine*

TF/IDF *Term Frequency/Inverse Document Frequency*

TP *True Positives*

Tree-LSTM *Tree Long Short Term Memory*

URIs *Uniform Resource Identifiers*

W3C *World Wide Web Consortium*

Sumário

1	Introdução	1
2	Referencial Teórico	6
2.1	A tarefa de <i>Perguntas e Respostas</i>	6
2.2	Grafos de Conhecimento	8
2.2.1	DBpedia	10
2.2.2	SPARQL	11
2.3	A tarefa de <i>Perguntas e Respostas em Grafos de Conhecimento</i>	13
2.4	Tree-LSTM	15
3	Trabalhos Relacionados	19
3.1	<i>Question Answering</i> (QA) em português	19
3.2	KGQA em português	22
3.3	KGQA em inglês	23
4	KGQA_{PT}: Um sistema de KGQA para o Português	26
4.1	Analisador Sintático	28
4.2	Classificador do Tipo de Pergunta	30
4.3	Mapeamento de Conceito	32
4.4	Geração de Consultas	38
4.5	Ranqueamento das Consultas	42
5	Avaliação Experimental	45

5.1	Datasets	45
5.2	Métricas de Avaliação	46
5.3	Resultados do Classificador do Tipo de Pergunta	47
5.4	Classificador de Relação Dupla	50
5.5	Tree-LSTM	51
5.6	KGQA _{PT}	52
6	Conclusão	57
	REFERÊNCIAS	59

1 Introdução

A produção de dados na era atual do *Big Data* é uma realidade impressionante e desafiadora. Estamos testemunhando uma explosão de dados, onde a capacidade de coletar, armazenar e analisar informações cresce exponencialmente. Um aspecto notável dessa produção de dados é a predominância de textos, oriundos de diversas fontes, como notícias online, *blogs*, enciclopédias virtuais e redes sociais ([TOPICS](#), s.d.). Embora os dados textuais sejam uma rica fonte de informação, eles apresentam desafios significativos, principalmente porque são predominantemente heterogêneos em conteúdo e escritos em linguagem natural. A linguagem natural é intrinsecamente não estruturada e ambígua, variando em contexto, estilo e significado. Tais características impõem complexidades na interpretação e processamento automático desses dados, exigindo abordagens sofisticadas para extrair significado e utilidade prática.

Uma das formas de extrair informações de fontes textuais é por meio da tarefa de QA tradicional ([MOMTAZI; ABBASIANAEI, 2022](#)). O objetivo dessa tarefa é responder automaticamente a perguntas formuladas em linguagem natural, baseando-se em fontes textuais. Por exemplo, dado um texto sobre a vida do piloto brasileiro de Fórmula 1 Ayrton Senna, um sistema de QA deve ser capaz de responder a perguntas tais como: “*Quando Ayrton Senna começou sua carreira?*” Tal abordagem representa um avanço significativo na interação homem-máquina, permitindo que usuários obtenham respostas específicas sem necessidade de pesquisar manualmente em documentos extensos. No entanto, a tarefa de QA tradicional enfrenta desafios intrínsecos à natureza dos textos, tais como a ambiguidade linguística e a variação de contextos. Para tratar esses problemas, os Grafos de Conhecimento (*Knowledge Graphs* (KGs)), surgiram como uma solução promissora ([MOMTAZI; ABBASIANAEI, 2022](#)), pois permitem estruturar informações e conhecimentos em forma de entidades e relações, proporcionando assim uma base mais concreta, computacionalmente eficiente e menos ambígua para o processamento de perguntas e respostas, entre outras tarefas. A tarefa de responder a consultas usando grafos

de conhecimento denomina-se KGQA, do inglês *Knowledge Graph Question Answering* (MOMTAZI; ABBASIANTEB, 2022).

A tarefa de KGQA representa um salto qualitativo em relação ao QA tradicional, pois utiliza KGs como fonte de informação, ajudando a superar muitas limitações do QA tradicional. Essa metodologia permite um processamento mais direto e preciso de perguntas, levando a respostas mais rápidas e confiáveis, dada a estruturação e organização prévia do conhecimento presente nos KGs. A pesquisa e implementação da tarefa de KGQA tem sido muito explorada em línguas com amplos recursos, como o inglês (SINGH, Kuldeep; RADHAKRISHNA, Arun Sethupat; BOTH, Andreas; SHEKARPOUR, Saeedeh; LYTRA, Ioanna; USBECK, Ricardo; VYAS, Akhilesh; KHIKMATULLAEV, Akmal; PUNJANI, Dharmen; LANGE, Christoph et al., 2018; RONY et al., 2022; PURKAYASTHA et al., 2022; TAN et al., 2023). No entanto, em línguas com menor produção de dados (em comparação com o inglês), como o português, essa área permanece relativamente inexplorada. Isso cria uma lacuna significativa na disponibilidade de ferramentas e soluções de KGQA adaptadas a essas linguagens. Portanto, a proposição de soluções de KGQA em português torna-se não apenas uma questão de equidade linguística, mas também um passo essencial para democratizar o acesso à informação e conhecimento.

Por outro lado, os modelos de linguagem pré-treinados alcançaram o estado da arte na tarefa de QA tradicional (JUN et al., 2022; YAMADA et al., 2020; YANG et al., 2019)¹. No entanto, ainda encontram dificuldades em responder perguntas que exigem aderência ao conhecimento factual. Os sistemas de KGQA geralmente convertem perguntas em linguagem natural para uma linguagem de consulta (MOMTAZI; ABBASIANTEB, 2022), por exemplo, SPARQL (W3C SEMANTIC WEB STANDARDS, 2023). Utilizando modelos de aprendizado de máquina, por exemplo, é possível converter o texto de uma pergunta em linguagem natural para uma consulta SPARQL e, assim, obter a resposta do *Knowledge Graph* (KG). Para treinar tais modelos, são utilizados conjuntos de dados abertos tais como QALD-7² e LCQuAD-v1³. O QALD-7 (USBECK et al., 2017) é um conjunto de dados multilíngua que inclui o português e fornece 215 pares de perguntas e suas correspondentes consultas SPARQL. Já o conjunto de dados LCQuAD-v1 (TRIVEDI et al., 2017) apresenta 5.000 perguntas complexas, somente em inglês, cada uma acompanhada de sua equivalente consulta SPARQL.

¹<https://paperswithcode.com/sota/question-answering-on-squad11>

²<https://github.com/ag-sc/QALD>

³<https://github.com/AskNowQA/LC-QuAD>

Estabelecer um mapeamento entre uma pergunta em linguagem natural e sua respectiva consulta SPARQL é um grande desafio, pois requer um alinhamento preciso entre os trechos do texto e as entidades e relações dentro do KG. As soluções conhecidas atualmente seguem geralmente duas abordagens: (i) treinar um modelo de Aprendizado de Máquina (AM) que traduz diretamente a pergunta em linguagem natural para uma linguagem de consulta, se valendo de modelos *sequence-to-sequence* (MOMTAZI; ABBASIANAE, 2022), ou (ii) dividir o processo de conversão em múltiplas etapas para reduzir a complexidade da tarefa (PURKAYASTHA et al., 2022). Enquanto a primeira abordagem exige métodos mais robustos e, conseqüentemente, mais exemplos de treinamento, a segunda demanda soluções eficazes para subproblemas, como a correta ligação entre entidades e relações da pergunta com os recursos corretos dentro do KG (MOMTAZI; ABBASIANAE, 2022). Dessa forma, muitas soluções de KGQA foram desenvolvidas principalmente para o inglês, beneficiando-se da abundância de recursos disponíveis para treinar modelos e da maturidade das tarefas subjacentes disponíveis nessa língua. Por outro lado, apenas um número limitado de iniciativas foi proposto para outros idiomas, como o português (MOMTAZI; ABBASIANAE, 2022).

Apesar do português ser a oitava língua mais falada no mundo, com mais de 263 milhões de falantes (EBERHARD; SIMONS; FENNIG, s.d.), e ser a quinta língua mais usada na Internet, com mais de 171 milhões de usuários (INTERNET WORLD STATS, s.d.), existem poucas iniciativas que abordam a tarefa de KGQA para o português. Para ilustrar isso, ao examinar as 76 soluções propostas para a tarefa de KGQA usando o conjunto de dados LcQuAD-v1 e o grafo de conhecimento DBPedia (AUER et al., 2007), apenas dez foram implementadas para idiomas além do inglês, e apenas uma é especificamente adaptada para o português⁴ (PEREVALOV et al., 2022). No caso do conjunto de dados QALD-9, que utiliza a DBPedia como grafo de conhecimento, todas as 49 soluções propostas eram exclusivamente para o inglês⁵ (PEREVALOV et al., 2022). Isso ressalta a necessidade de desenvolver soluções de KGQA para outras línguas além do inglês, incluindo o português.

Além disso, muitos idiomas apresentam desafios únicos. Um exemplo ilustrativo que destaca a dificuldade em converter uma pergunta do português para SPARQL é a conjugação de verbos. O português apresenta uma multiplicidade de tempos verbais, cada um com formas de conjugação distintas para diferentes pessoas e números, o que pode introduzir complexidade ao tentar traduzir diretamente uma pergunta escrita em português

⁴<https://github.com/KGQA/leaderboard/blob/gh-pages/dbpedia/lcquad.md#lc-quad-v1>

⁵<https://github.com/KGQA/leaderboard/blob/gh-pages/dbpedia/qald.md>

para uma consulta SPARQL. Considere, por exemplo, a pergunta em português: “*Quais filmes foram dirigidos por Quentin Tarantino?*”. Ao traduzir isso para SPARQL, o desafio reside em identificar a propriedade apropriada dentro do KG para o trecho “*dirigidos por*”. Essa complexidade surge das inúmeras possíveis conjugações do verbo “*dirigir*” em português. Esse exemplo também ilustra outro problema comum não só para o português, como para várias línguas: a polissemia, quando uma mesma palavra pode assumir diferentes significados a depender do contexto. No caso mencionado, a palavra “*dirigir*” pode assumir diferentes significados em português, como “*dirigir um filme*”, “*dirigir um carro*” ou “*dirigir uma empresa*”, tornando a tarefa de KGQA em português ainda mais desafiadora.

É importante destacar ainda que treinar modelos de AM utilizando conjuntos de dados com um número limitado de exemplos, como visto no caso do QALD-7 com apenas 215 exemplos de treinamento, apresenta desafios substanciais. O principal obstáculo surge da insuficiente variedade e diversidade nos dados de treinamento. Quando os exemplos são escassos, o modelo pode enfrentar dificuldades em aprender padrões e aplicar esse conhecimento a novos exemplos, reduzindo assim sua eficácia em aplicações práticas, como traduzir uma pergunta em linguagem natural para uma consulta SPARQL.

Este trabalho propõe uma abordagem para a tarefa de KGQA em português, chamada de KGQA_{PT}⁶. A abordagem proposta organiza o processo de conversão em cinco componentes: (i) Analisador Sintático, (ii) Classificação do Tipo de Pergunta, (iii) Mapeamento de Conceitos, (iv) Geração de Consultas e (v) Ranqueamento de Consultas. Considere, por exemplo, a seguinte pergunta de entrada: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*”. Primeiro, o componente de Analisador Sintático extrai características sintáticas, como Tokenização, POS-Tagging, Lematização, Árvore de Dependência Sintática e Stemização. Em seguida, o componente de Classificação do Tipo de Pergunta categoriza a pergunta em um dos três tipos possíveis: Booleano, Contagem ou Lista. No caso da pergunta mencionada, a classificação seria do tipo “Lista”. O componente de Mapeamento de Conceitos lida com a Ligação de Entidades, vinculando o termo “Ayrton Senna” ao recurso da DBPedia “`dbr:Ayrton_Senna`” e o trecho “Fórmula 1” ao recurso “`dbr:Formula_One`”. Ele também realiza a Ligação de Relações, associando o termo “campeão” à propriedade “`dbp:champions`”.

Com base nas informações dos componentes anteriores, o componente de Geração de Consultas gera uma lista de consultas candidatas na linguagem SPARQL. Então, o com-

⁶<https://github.com/elbemiranda/KGQApt>

ponente de Classificação de Consultas organiza essas consultas de acordo com critérios de similaridade, selecionando finalmente a consulta de maior classificação como resposta. No nosso exemplo, a consulta escolhida é: “SELECT ?resp WHERE ?resp dbp:champions dbr:Ayrton_Senna”. A Figura 9 ilustra o método proposto com base no exemplo fornecido. O sistema proposto foi avaliado utilizando os datasets padrão QALD e LCQuAD, demonstrando bons resultados com um F1-score de até 41.9% no QALD e de 45.8% no LCQuAD. As contribuições deste trabalho também incluem modelos treinados em português para a tarefa de KGQA, um novo método de Ligação de Relações para o português chamado PTRL, além de estabelecer um baseline para a tarefa de KGQA em português.

Neste trabalho serão respondidas as seguintes questões de pesquisa:

1. Qual é o desempenho de um sistema baseado em componentes que aborda a tarefa de KGQA para a língua portuguesa?
2. Qual é o impacto dos componentes do modelo no seu desempenho?

Este trabalho está organizado da seguinte forma: o Capítulo 2 revisa brevemente os conceitos-chave subjacentes à tarefa de KGQA, enquanto o Capítulo 3 apresenta trabalhos relacionados. Já o Capítulo 4 apresenta a solução desenvolvida para resolver a tarefa de KGQA em português. O Capítulo 5 traz os resultados experimentais, e as conclusões são abordadas no Capítulo 6.

2 Referencial Teórico

2.1 A tarefa de *Perguntas e Respostas*

A tarefa de Perguntas e Respostas (QA) é conhecida e explorada há bastante tempo. Tradicionalmente envolve responder a uma pergunta feita em linguagem natural utilizando como referência uma base de conhecimento que pode conter um ou vários textos de entrada também escritos em linguagem natural ([MOMTAZI; ABBASIANAEI, 2022](#)). Considere, por exemplo, a seguinte pergunta: *Quando Ayrton Senna começou a sua carreira?*. Um sistema de QA deveria ser capaz de responder corretamente o ano de 1973, dado o contexto abaixo.

Pergunta: Quando Ayrton Senna começou sua carreira?

Contexto: Ayrton Senna da Silva foi um piloto de Fórmula 1, empresário e filantropo brasileiro. Senna foi campeão da categoria de piloto três vezes, em 1988, 1990 e 1991. Começou sua carreira competindo no kart em 1973 e em carros de fórmula em 1981, quando venceu as Fórmulas Ford 1600 e 2000.

Resposta: 1973

Geralmente, um sistema de QA consiste basicamente de três módulos: análise da pergunta, busca em documentos e extração da resposta ([THAMBI; REGHURAJ, 2022](#)). A etapa de análise da pergunta envolve identificar o tipo de pergunta e também o tipo de resposta desejada, que podem ser vários, tais como factóide, hipotético, booleano, causal ou contagem. Para a identificação do tipo de pergunta são aplicadas ao texto da pergunta uma série de tarefas de pré-processamento para extrair característica sintáticas e morfológicas, como por exemplo, Tokenização, Lemmatização, POS *Tagging*, etc. De posse das características sintáticas, a pergunta é estruturada em uma consulta que é submetida à etapa seguinte de busca em documentos.

Vários métodos podem ser aplicados durante a etapa de busca em documentos, como por exemplo, o método algébrico que usa o conceito de vetores, matrizes e tuplas, o método de teoria dos conjuntos que trata os documentos como um conjunto de palavras ou frases e os métodos probabilísticos baseados em TF/IDF ([MOMTAZI; ABBASIANTEB, 2022](#)). Ao fim da etapa de busca em documentos, um conjunto de possíveis respostas é selecionado e passa por um processo de ranqueamento, onde as respostas são ordenadas por relevância com base em uma nota. Tal procedimento permite executar primeiro as consultas com maior probabilidade de retornar a resposta correta, segundo algum critério de relevância. A última etapa do processo é a de extração da resposta onde ocorre o processamento dos documentos candidatos que contém as potenciais respostas. As respostas podem ser dadas em diversos formatos, como por exemplo, um documento ou frase contendo a resposta ou mesmo as próprias palavras que representam a resposta correta.

Embora a tarefa de perguntas e respostas tenha tido grandes avanços nos últimos anos, diversos desafios ainda encontram-se em aberto ([THAMBI; REGHURAJ, 2022](#)), tais como:

- Dificuldade de entender uma pergunta em linguagem natural, já que uma mesma pergunta pode ser formuladas de diversas formas e pode conter erros de grafia e concordância.
- Dificuldade de extrair conhecimento de fontes de dados não-estruturadas como texto livre, além de ter que lidar com ambiguidades e correferências.
- Dificuldade de capturar informações contextuais durante o processo de consulta, tornando difícil reduzir ambiguidades.

Muitos dos problemas citados acima surgem do fato de textos livres serem fontes não estruturadas de informação. Como uma alternativa para resolver esse problema, poderíamos utilizar fontes estruturadas de informação, tais como base de dados relacionais. Entretanto, a construção de tais bancos de dados exige um esforço manual muito grande.

Vale ressaltar que com o avanço da área de *Deep Learning* (DL), várias abordagens têm surgido com bons resultados, alcançando o estado da arte na tarefa de QA, tais como ANNA ([JUN et al., 2022](#)) com um F1 de 95.7%, LUKE ([YAMADA et al., 2020](#)) com F1 de 95.4% e XLNET ([YANG et al., 2019](#)) com F1 de 95.1%, entre outros. Alguns desses métodos representam um documento como um vetor semântico e as respostas são obtidas através de um cálculo de similaridade entre o vetor de pergunta e os vetores de possíveis

respostas. Embora os modelos de linguagem pré-treinados tenham alcançado o estado da arte na tarefa de QA, ainda encontram dificuldades em responder perguntas que exigem aderência ao conhecimento factual (MOMTAZI; ABBASIANTEB, 2022).

A tarefa de *Knowledge Graph Question Answering* (KGQA) é um método de perguntas e respostas que tem crescido muito nos últimos anos e utiliza fontes estruturadas de informação conhecidos como Grafos de Conhecimento em vez de texto livre. Antes de detalharmos o funcionamento da tarefa de KGQA, iremos antes entender o que são Grafos de Conhecimento.

2.2 Grafos de Conhecimento

Existem diversas definições sobre o que é um Grafo de Conhecimento (GC). Neste trabalho, um GC é uma estrutura de dados formada por nós, que representam entidades, e arestas, que ligam os nós entre si e representam relações entre as entidades (HOGAN et al., 2020). As entidades em um GC podem representar pessoas, lugares, objetos, etc. As relações, por sua vez, pode simbolizar, por exemplo, o local de nascimento e a data de nascimento de uma pessoa. A informação é armazenada no grafo de conhecimento no formato de fatos, onde cada fato é formado por uma tripla indicando a relação entre duas entidades (MOMTAZI; ABBASIANTEB, 2022). Portanto, um GC é uma estrutura de dados $KG = (V, E, R)$ onde V é o conjunto de entidades, R é o conjunto de tipos de relações, e $E = (h, r, t)$ é uma aresta representando um fato, com $h, t \in V$, também chamados de sujeito/objeto, ou *head/tail*, e $r \in R$, também chamado de predicado. Os KGs fornecem uma representação estruturada das relações semânticas entre entidades no mundo real. Suponha que se queira representar em um KG a seguinte pergunta “*Em qual campeonato de Fórmula 1 Ayrton Senna foi campeão?*”. A resposta seria um subgrafo $KG' = (V', E', R')$, onde $KG' \subset KG$. A Figura 1 exibe um diagrama que ilustra KG' .

Os grafos de conhecimento têm se mostrado uma maneira eficiente de guardar, buscar e recuperar informação (ABU-SALIH, 2021). Eles podem conter desde fatos muito simples até muito complexos. **Fatos simples** são aqueles representados por uma única tripla, ou seja, apenas dois nós estão ligados entre si por uma aresta. Como exemplo de um fato simples, temos a seguinte frase afirmativa: “*Ayrton Senna nasceu em São Paulo*”. Tal fato pode ser representado em um grafo de conhecimento através de uma única tripla, contendo as entidades “*Ayrton Senna*” (“*dbr:Ayrton_Senna*” na DBpedia) e “*São Paulo*”

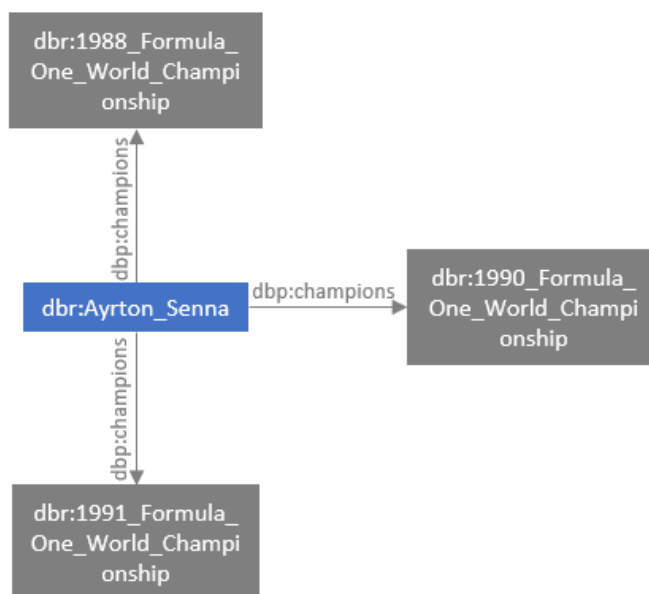


Figura 1: Exemplo de um subgrafo do Grafo de Conhecimento DBpedia.

(“dbr:São_Paulo” na DBpedia) e uma relação entre elas “*nasceu em*” (“dbo:birthPlace” na DBpedia), conforme representado na Figura 2.

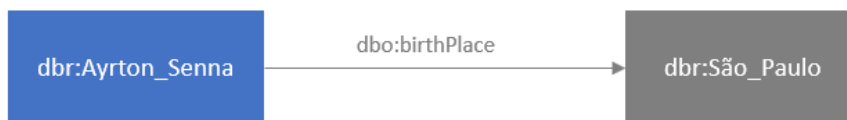


Figura 2: Grafo representando o fato simples: “*Ayrton Senna nasceu em São Paulo*”.

Fatos complexos são aqueles cuja representação exige mais de uma tripla. Para um exemplo de um fato complexo, poderíamos ter a seguinte frase: “*Ayrton Senna foi o vencedor do GP de Mônaco, ocorrido em 1993*”. Dessa frase, podemos extrair as entidades “*Ayrton Senna*” (“dbr:Ayrton_Senna” na DBpedia), “*GP de Mônaco*” (“dbr:1993_Monaco_Grand_Prix” na DBpedia) e “*1993*”. Além disso, também podemos identificar a relação “*vencedor de*” (“dbr:firstDriver” na DBpedia), que liga as entidades “dbr:Ayrton_Senna” e “dbr:1993_Monaco_Grand_Prix”, e a relação “*ocorreu em*” (“dbp:year” na DBpedia) ligando a entidade “dbr:1993_Monaco_Grand_Prix” e o número inteiro “1993”. A Figura 3 ilustra um exemplo de grafo que representa o fato complexo exemplificado.

Os grafos de conhecimento geralmente são representados usando o formato *Resource Description Framework* (RDF). **RDF** é um formato para dados de grafos orientados e rotulados que armazena fatos na forma de triplas (s,p,o), onde s é o sujeito, p é o



Figura 3: Grafo representando o fato complexo: “Ayrton Senna foi o vencedor do GP de Mônaco, ocorrido em 1993”

predicado e *o* é o objeto. Os sujeitos são representados por recursos e são identificados usando *Uniform Resource Identifiers* (URIs), que são identificadores únicos que podem representar entidades do mundo real, conceitos abstratos, documentos e etc. Predicados representam as propriedades ou atributos desses recursos, enquanto objetos representam os valores dessas propriedades ou as relações com outros recursos.

Podemos dividir os grafos de conhecimento de acordo com o escopo e abrangência. Por exemplo, eles podem ser de domínio aberto ou de domínio específico. Os GC de domínio aberto contêm muitos fatos de diversos assuntos e temas, os mais populares desse tipo são a DBPedia ([auer2007dbpedia](#)), Freebase ([BOLLACKER et al., 2008](#)), Yago ([SUCHANEK; KASNECI; WEIKUM, 2007](#)), Wikidata ([VRANDEČIĆ; KRÖTZSCH, 2014](#)) e o Google Knowledge Graph ([SINGHAL, 2012](#)). Os grafos de conhecimento de domínio específico, como o próprio nome já sugere, são criados para um domínio bem restrito. Temos como exemplos, o Auto Insurance KG ([ZHANG; WU et al., 2021](#)), um GC de seguros de carros construído para ajudar na prevenção de fraude de seguros, o Microsoft Academic Graph ([FÄRBER, 2019](#)), um grafo de conhecimento de dados acadêmicos, MusicBrainz ([BERTRAM; DUNKEL; HERMOSO, 2023](#)) um GC do domínio da música, apenas para citar alguns.

2.2.1 DBpedia

A DBpedia é um KG derivado de fatos contidos nos milhões de artigos da enciclopédia virtual Wikipedia¹ ([AUER et al., 2007](#)). Seu objetivo é prover uma forma de consultar os dados da Wikipedia de uma maneira estruturada. Por exemplo, a página da Wikipedia sobre Ayrton Senna² é representada na DBpedia como um nó, com arestas conectando-a a outros nós que representam informações como data de nascimento e realizações. Tais informações são extraídas das caixas de informações da página do Ayrton Senna na Wikipedia, conforme pode ser visto à direita da Figura 4. Os dados das caixas de informações

¹<https://www.wikipedia.org/>

²https://pt.wikipedia.org/wiki/Ayrton_Senna

são extraídos em formato de triplas, onde as entidades e relações são mapeadas de acordo com a ontologia da DBpedia, como pode ser visualizado na Figura 5. Todas as triplas extraídas das caixas de informações são importadas para um banco de dados semântico que possibilita a execução de consultas através da linguagem SPARQL.

Ayrton Senna

Artigo

Discussão

Ler

Editar

Ver histórico

Ferramentas

Origem: Wikipédia, a enciclopédia livre.

★

🔍

Nota: Para outros significados, veja [Ayrton Senna \(desambiguação\)](#).

Ayrton Senna da Silva (ONM • ComRB • CvMA • OME) (São Paulo, 21 de março de 1960 — Bolonha, 1 de maio de 1994) foi um piloto de Fórmula 1, empresário e filantropo brasileiro. Senna foi campeão da categoria de piloto três vezes, em 1988, 1990 e 1991. Começou sua carreira competindo no [kart](#) em 1973 e em "carros de fórmula" em 1981, quando venceu as [Fórmulas Ford](#) 1600 e 2000. Em 1983 alcançou o título de campeão do [Campeonato Britânico de Fórmula 3](#) batendo vários recordes. Seu desempenho impulsionou sua ascensão à Fórmula 1, fazendo sua primeira aparição na categoria no [Grande Prêmio do Brasil de 1984](#) pela equipe [Toleman-Hart](#). Em sua primeira temporada, Senna pontuou em cinco corridas, fechando o ano com treze pontos e a nona posição na classificação geral dos pilotos. No ano seguinte, ingressou na [Lotus-Renault](#), pela qual venceu seis grandes prêmios ao longo de três temporadas.

Em 1988, juntou-se ao francês [Alain Prost](#) na [McLaren-Honda](#), com o qual teve grande rivalidade. Senna venceu oito etapas daquela temporada e sagrou-se campeão mundial pela primeira vez. Após a polêmica final de 1989 com Prost que resultou na segunda colocação do torneio, ele retomou o título em 1990, vencendo novamente na temporada seguinte, tornando-se o piloto mais jovem a conquistar um tricampeonato na Fórmula 1 até então. Em 1993, Senna foi vice-campeão, vencendo cinco corridas. Transferiu-se para a [Williams](#) em 1994, onde disputou apenas três etapas, a última sendo o [Grande Prêmio de San Marino](#), onde se [acidentou e morreu](#). Ao todo, Senna participou de 161 grandes prêmios na Fórmula 1, alcançando **41 vitórias**, 80 pódios, **65 pole positions** e 19 voltas mais rápidas.

Além das corridas de carros, Senna dedicava-se a [jet skis](#), [motos](#), [aeromodelos](#) e principalmente [helicópteros](#). Ele também administrava diversas [marcas](#) e [empreendimentos](#), além de ter [patrocinado](#) vários programas de [assistência filantrópica](#), principalmente os ligados a crianças carentes. Depois de morrer, sua irmã, [Viviane Senna](#), fundou o [Instituto Ayrton Senna](#), uma organização não governamental que oferece oportunidades de desenvolvimento humano a crianças e jovens de baixa renda. Além disso, o personagem [Senninha](#) foi criado com a intenção de atingir o público infantil com os ideais do piloto, como a superação, dedicação e o gosto pela vitória.

Sua [morte](#), assim como o funeral e velório, provocou uma das maiores comoções da história do

Ayrton Senna



Informações pessoais

Nome completo

Ayrton Senna da Silva

Apelido(s)

"Beco"^[1]
"O Chefe"^[2]
"O Rei de Mônaco"^[1]
"Magic Senna"^[3]
"Rei da Chuva"^[4]

Nacionalidade

brasileiro

Nascimento

21 de março de 1960
São Paulo, SP, Brasil

Morte

1 de maio de 1994 (34 anos)
Bolonha, Emilia-Romanha, Itália

Causa da morte

acidente automobilístico

Altura

1,76 m


Parentesco

Viviane Senna (irmã)
Leonardo Senna (irmão)
Bruno Senna (sobrinho)

Figura 4: Página do Ayrton Senna na Wikipedia.

2.2.2 SPARQL


SPARQL é uma linguagem de consulta padronizada projetada para recuperar informações de fontes de dados RDF. Ela permite consultar grafos de conhecimento que aderem ao modelo RDF, facilitando a recuperação de dados e o acesso a informações estruturadas. SPARQL permite que os usuários realizem consultas complexas em grafos RDF, combinando critérios de pesquisa, filtragem, junção e agregação. A linguagem suporta consultas baseadas em padrões de triplas, baseadas em variáveis, expressões condicionais, filtros, etc. SPARQL é amplamente adotado e padronizado pelo *World Wide Web Consortium*


Browse using
Formats
Faceted Browser
Sparql Endpoint

About: [Ayrton Senna](#)

An Entity of Type: [animal](#), from Named Graph: <http://dbpedia.org>, within Data Space: [dbpedia.org](#)

Ayrton Senna da Silva ONM • ComRB • CvMA • OME (São Paulo, 21 de março de 1960 — Bolonha, 1 de maio de 1994) foi um piloto brasileiro de Fórmula 1, campeão da categoria três vezes, em 1988, 1990 e 1991. Ele começou sua carreira competindo no kart em 1973 e em "carros de fórmula" em 1981, quando venceu as Fórmulas Ford 1600 e 2000. Em 1983 alcançou o título de campeão do Campeonato Britânico de Fórmula 3 batendo vários recordes. Seu desempenho impulsionou sua ascensão à Fórmula 1, fazendo sua primeira aparição na categoria no Grande Prêmio do Brasil de 1984 pela equipe Toleman-Hart. Em sua primeira temporada, Senna pontuou em cinco corridas, fechando o ano com treze pontos e a nona posição na classificação geral dos pilotos. No ano seguinte, ingressou na Lotus-Renault, pela qual venceu seis



Property	Value
dbo:birthDate	<ul style="list-style-type: none"> 1960-03-21 (xsd:date)
dbo:birthName	<ul style="list-style-type: none"> Ayrton Senna da Silva (en)
dbo:birthPlace	<ul style="list-style-type: none"> dbr:1986_Brazilian_Grand_Prix dbr:São_Paulo dbr:São_Paulo_(state)
dbo:birthYear	<ul style="list-style-type: none"> 1960-01-01 (xsd:gYear)
dbo:deathDate	<ul style="list-style-type: none"> 1994-05-01 (xsd:date)
dbo:deathPlace	<ul style="list-style-type: none"> dbr:Bologna dbr:1985_Italian_Grand_Prix dbr:Emilia-Romagna
dbo:deathYear	<ul style="list-style-type: none"> 1994-01-01 (xsd:gYear)

Figura 5: Página do Ayrton Senna na DBpedia.

(W3C) ([W3C SEMANTIC WEB STANDARDS, 2023](#)), servindo como uma tecnologia fundamental para acessar e explorar dados em KGs baseados em RDF.

Considerando que um KG é representado usando RDF e consultado usando SPARQL, a tarefa de KGQA busca responder perguntas feitas em uma linguagem natural Q_{NL} , com base nas informações presentes em um grafo de conhecimento KG . O processo inclui a conversão da pergunta de entrada Q_{NL} em uma consulta formal Q_{FL} , usando uma função de transformação F_{KGQA} , onde $F_{KGQA} : Q_{NL} \rightarrow Q_{FL}$. A maioria dos sistemas de KGQA visa desenvolver modelos que aprendam eficientemente a função F_{KGQA} . No Exemplo 4.3 podemos visualizar qual seria uma consulta SPARQL que responde corretamente à pergunta “Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?”

2.3 A tarefa de Perguntas e Respostas em Grafos de Conhecimento

Como destacado no final da seção 2.1, a tarefa de KGQA surge com uma vantajosa alternativa para a tarefa de *Question Answering* tradicional, visto que os GC são fontes de dados estruturados, o que elimina muitos dos problemas descritos anteriormente. As etapas para a resolução da tarefa de KGQA são muito similares às realizadas pela tarefa de QA tradicional. Consistem basicamente em: Ligação de Entidade (LE) (*Entity Linking* (EL)), Ligação de Relação (LR) (*Relation Linking* (RL)), Construção da Consulta (CC) (*Query Building* (QB)), e Ranqueamento de Consulta (RC) (*Query Ranking* (QR)).

Após a etapa de construção das consultas, elas são submetidas ao grafo de conhecimento, que retornará com possíveis respostas. Essas respostas podem estar a um salto da entidade identificada (*single-hop*) ou a vários saltos (*multi-hop*). No exemplo da Figura 6, para a pergunta “Onde nasceu o piloto Ayrton Senna?” a resposta está a um salto de distância, ou seja, o nó que contém a resposta “dbr:São_Paulo” está diretamente ligado através da relação “dbo:birthPlace” ao nó da entidade “dbr:Ayrton_Senna”.

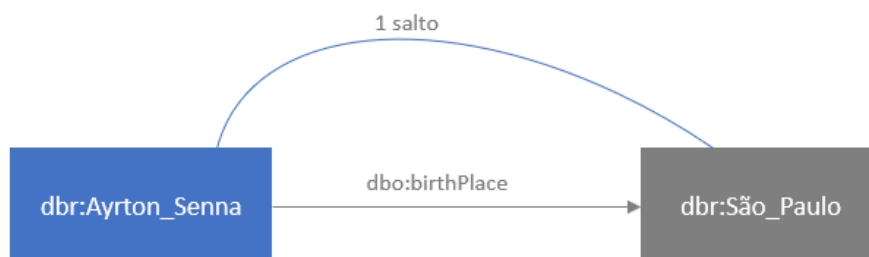


Figura 6: A resposta à pergunta “Onde nasceu o piloto Ayrton Senna?” está a um salto de distância da entidade “dbr:Ayrton_Senna” identificada na pergunta.

Suponha agora que queremos responder à pergunta: “Quem foi o diretor do documentário de Ayrton Senna?”, considerando o KG da Figura 7. Nesse caso, a resposta correta estará a dois saltos de distância; no primeiro salto, identificamos a relação “dbo:starring” entre o nó “dbr:Ayrton_Senna” e o nó “dbr:Senna_(film)” e, após isso, identificamos a relação “dbo:director” entre o nó “dbr:Senna_(film)” e o nó “dbr:Asif_Kapadia”, que é a resposta à pergunta.

Dentre as principais técnicas de KGQA, podemos destacar as baseadas em Analisador Semântico (*Semantic Parser* (SP)) ou em Recuperação da Informação (*Information Retri-*



Figura 7: A resposta à pergunta: “*Quem foi o diretor do documentário de Ayrton Senna?*” está a dois saltos de distância do nó “**dbr:Ayrton_Senna**” identificado na pergunta.

eval (IR)) e, ainda, os métodos baseados em algoritmos de Redes Neurais de Aprendizado Profundo.

Nos últimos anos, muitos métodos têm sido desenvolvidos para resolver a tarefa de KGQA em diversas línguas, mas como acontece com a maioria das iniciativas na área de Processamento de Linguagem Natural (PLN) (*Natural Language Processing* (NLP)), a grande concentração das soluções propostas está focada no inglês. Duas das subtarefas mais importantes para a tarefa de KGQA são a Ligação de Entidade e a Ligação de Relação.

A tarefa de **LE** envolve extrair corretamente de um texto as menções de pessoas, empresas, lugares, datas, etc. e relacioná-las corretamente com as entidades presentes em um grafo de conhecimento. Por exemplo, na pergunta: “*Onde nasceu o piloto Ayrton Senna?*”, a tarefa de LE deve ser capaz relacionar o trecho “*Ayrton Senna*” com a entidade “**dbr:Ayrton_Senna**” da DBpedia. Vários sistemas já foram desenvolvidos para resolver a tarefa de LE para o inglês com excelentes desempenhos, como por exemplo: SPEL (SHAVARANI; SARKAR, 2023), ENTQA (ZHANG; HUA; STRATOS, 2022), REFINED (AYOOLA et al., 2022) e CHOLAN (RAVI et al., 2021), dentre vários outros. Já para o português, foram encontrados apenas dois sistemas que realizam a tarefa de LE: DBPEDIA SPOTLIGHT (DAIBER et al., 2013; MENDES et al., 2011) e WIKIFIER (BRANK; LEBAN; GROBELNIK, 2017).

Já a tarefa de **LR** consiste em extrair as relações contidas em um texto e classificá-las corretamente nos tipos corretos de arestas de um grafo de conhecimento. Ainda tomando como exemplo a frase “*Onde nasceu o piloto Ayrton Senna?*”, a tarefa de LR deveria corretamente relacionar o texto “*nasceu*” com a propriedade “**dbo:birthPlace**” da DBpedia. Mesmo para o inglês, os modelos para LR são mais escassos, dentre eles podemos destacar: EARL (DUBEY et al., 2018), FALCON (SAKOR; SINGH; VIDAL, 2019) e RNLIWOD (SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018). Destes, apenas o RNLIWOD

pôde ser utilizado para o português, através da adaptação de alguns de seus componentes para lidar com as especificidades do português, como adaptação do *Stemmer* e tradução para o português do dicionário de relações utilizado pelo sistema.

Resolver bem as subtarefas LE e LR é crucial para um bom funcionamento de um sistema de KGQA, pois uma identificação incorreta de uma entidade ou de uma relação pode inviabilizar o sucesso na busca do nó de resposta dentro do GC, isso ocorre porque o nó de resposta no grafo de conhecimento está direta ou indiretamente ligado (através da relação identificada pela LR) ao nó identificado pela LE.

2.4 Tree-LSTM

As *Recurrent Neural Networks* (RNNs) são um tipo de arquitetura de redes neurais projetadas para lidar com sequências de dados, como séries temporais ou texto. Ao contrário das redes neurais tradicionais, que processam cada entrada de forma independente, as RNNs possuem conexões recorrentes, o que lhes permite manter uma memória interna e capturar informações contextuais ao longo do tempo. Uma RNN é capaz de processar sua entrada através da aplicação de uma função de transição h_t . A cada período de tempo t a função h_t recebe um vetor x_t e o estado anterior h_{t-1} . A função h_t pode ser definida como:

$$h_t = \tanh(W_{x_t} + U_{h_{t-1}} + b) \quad (2.1)$$

O problema com a RNN é que, durante o treinamento, o seu gradiente pode crescer ou decrescer exponencialmente quando lidamos com sentenças muito longas. Para resolver esse problema, surgiram as *Long Short Term Memory* (LSTM), que introduziram uma célula de memória que ajuda a preservar o seu estado por um longo período de tempo (ZAREMBA; SUTSKEVER, 2015). Dado um intervalo de tempo t , um *gate* de entrada i_t , um *gate* de saída o_t , uma célula de memória c_t , um estado oculto h_t e x_t representando a entrada no tempo t , uma LSTM pode ser definida como:

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}), \quad (2.2)$$

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}), \quad (2.3)$$

$$o_t = \sigma \left(W^{(o)} x_t + U^{(o)} h_{t-1} + b^{(o)} \right), \quad (2.4)$$

$$u_t = \tanh \left(W^{(u)} x_t + U^{(u)} h_{t-1} + b^{(u)} \right), \quad (2.5)$$

$$c_t = i_t \odot u_t + f_t \odot c_{t-1}, \quad (2.6)$$

$$h_t = o_t \odot \tanh(c_t), \quad (2.7)$$

onde σ é uma função sigmóide logística e \odot representa uma multiplicação elemento a elemento. A intuição por trás dessas fórmulas é que f_t controla o quanto será esquecido em relação à célula de memória anterior c_{t-1} . Por sua vez, i_t controla o quanto cada célula é atualizada e o_t controla o quanto do estado de memória da célula atual será exposto para as próximas (ZAREMBA; SUTSKEVER, 2015). Na Figura 8 podemos visualizar um diagrama ilustrativo de uma LSTM. Uma das limitações da LSTM é não possibilitar agregar informação em nível hierárquico, mas somente sequencial.

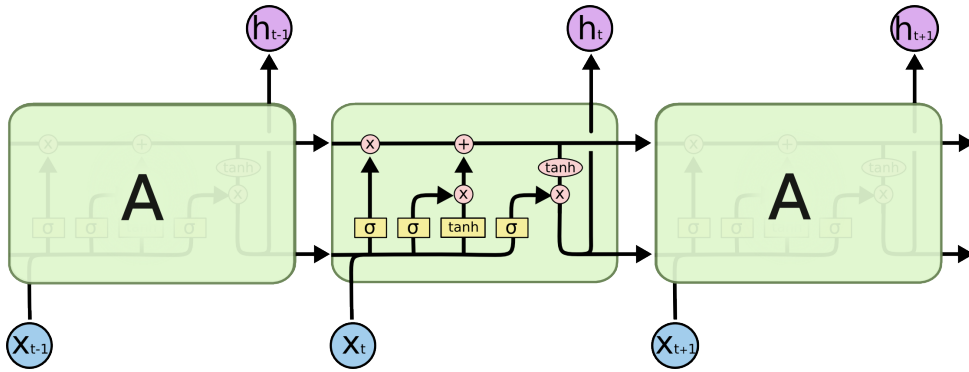


Figura 8: Diagrama ilustrativo de uma LSTM. (OLAH, s.d.)

Um modelo Tree-LSTM é uma generalização do modelo LSTM para uma arquitetura baseada em árvore. Enquanto em um modelo LSTM, o seu estado é computado com base na entrada atual e no estado das unidades anteriores, uma Tree-LSTM tem seu estado calculado com base nos estados de tantas quantas unidades filhas tiver. Uma LSTM pode ser considerada um caso particular de uma Tree-LSTM, onde cada nó possui exatamente um nó filho (TAI; SOCHER; MANNING, 2015). Um modelo baseado em árvore é especialmente atrativo especialmente quando estamos lidando com texto, dada a hierarquia da estrutura sintática de uma sentença. Em (TAI; SOCHER; MANNING, 2015), os autores demonstram que o modelo de Tree-LSTM supera os modelos sequenciais

estado da arte em duas tarefas: análise de sentimento e predição de relação semântica entre duas sentenças.

Similar a uma LSTM, uma Tree-LSTM também tem um *gate* de entrada i_j , um *gate* de saída o_j , uma célula de memória c_j e um estado oculto h_j . A grande diferença é que os portões de entrada e saída e a célula de memória são dependentes dos estados das unidades filhas. Em vez de apenas um *gate* de esquecimento, uma Tree-LSTM tem um *gate* de esquecimento f_{jk} para cada filha k e é isso que permite a Tree-LSTM incorporar informações dos nós filhos. Dados os portões i_j , o_j , f_{jk} , a célula c_j , o estado h_j , um vetor de entrada x_t representando uma palavra da sentença e $C(j)$ um conjunto de nós filhos do nó j , podemos representar uma Tree-LSTM como:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k, \quad (2.8)$$

$$i_j = \sigma \left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)} \right), \quad (2.9)$$

$$f_{jk} = \sigma \left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)} \right), \text{ onde } k \in C(j) \quad (2.10)$$

$$o_j = \sigma \left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)} \right), \quad (2.11)$$

$$u_j = \tanh \left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)} \right), \quad (2.12)$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \quad (2.13)$$

$$h_j = o_j \odot \tanh(c_j) \quad (2.14)$$

Podemos utilizar o modelo Tree-LSTM para calcular o grau de similaridade entre duas representações em árvore de duas sentenças. O objetivo é predizer um grau de similaridade que está em um intervalo $[1, K]$, $K > 1$. Primeiro, obtemos as representações vetoriais h_L e h_R das duas sentenças passando suas árvores de dependência sintática para uma Tree-LSTM. A partir daí, podemos predizer o grau de similaridade \hat{y} utilizando uma

Rede Neural que considera tanto a distância quanto o ângulo entre os vetores h_L e h_R (TAI; SOCHER; MANNING, 2015):

$$h_{\times} = h_L \odot h_R, \quad (2.15)$$

$$h_{+} = |h_L - h_R|, \quad (2.16)$$

$$h_s = \sigma \left(W^{(\times)} h_{\times} + W^{(+)} h_{+} + b^{(h)} \right), \quad (2.17)$$

$$\hat{p}_{\theta} = \text{softmax} \left(W^{(p)} h_s + b^{(p)} \right), \quad (2.18)$$

$$\hat{y} = r^T \hat{p}_{\theta}, \quad (2.19)$$

onde $r^T = [1, 2, \dots, K]$ e a função de valor absoluto são aplicados elemento a elemento. O motivo do uso de h_{\times} e h_{+} é empírico, uma vez que notou-se que o desempenho da combinação dos dois era melhor que o uso de cada um separadamente (TAI; SOCHER; MANNING, 2015). Embora fosse possível utilizar uma função de custo tradicional tal como a função dos mínimos quadrados, a função de custo a seguir retornou melhores resultados (TAI; SOCHER; MANNING, 2015). Primeiramente é definida uma distribuição alvo p que satisfaça a restrição de $y = r^T p$:

$$p_i = \begin{cases} y - \lfloor y \rfloor, & i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & i = \lfloor y \rfloor \\ 0 & \text{caso contrário} \end{cases} \quad (2.20)$$

onde $1 \leq i \leq K$. A função de custo utilizada é a divergência de Kullback-Leibler (KL) regularizada entre p e \hat{p}_{θ} :

$$J(\theta) = \frac{1}{m} \sum_{k=1}^m KL \left(p^{(k)} \parallel \hat{p}_{\theta}^{(k)} \right) + \frac{\lambda}{2} \|\theta\|_2^2, \quad (2.21)$$

onde m é a quantidade de pares de sentenças para o treinamento e k é o índice do par de sentenças.

3 Trabalhos Relacionados

3.1 QA em português

Alguns trabalhos têm sido desenvolvidos com o objetivo de criar novos recursos para o português, bem como tentar resolver as tarefas de QA e KGQA em português. Em (SANTOS; ROCHA, 2005) os autores discutem o trabalho realizado para incluir o português nas tarefas e competições da *Conference and Labs of the Evaluation Forum* (CLEF). Foram criados vários recursos para o português que compõem a LINGUATECA¹, um conjunto de recursos de processamento de linguagem natural para o português. Dentre os recursos produzidos, foi disponibilizado o corpus anotado CETEMPÚBLICO² e a FLORESTA SINTÁ(C)TICA³, além da organização das competições de avaliação de recuperação da informação e de QA voltadas para o português. Além disso, foi disponibilizado um *data-set* incluindo a criação de 100 perguntas em linguagem natural em português com suas respectivas respostas, a tradução dessas perguntas para o inglês e também a tradução de 600 outras perguntas de inglês para português. Os autores também relatam o processo de criação da coleção CHAVE⁴, utilizando textos do jornal PÚBLICO⁵, superando vários desafios técnicos, como a adaptação do formato dos textos e a limpeza manual de alguns documentos.

Tendo como base a tarefa de QA tradicional, (AMARAL et al., 2006) desenvolveram um sistema de QA para o português que emprega tecnologias de processamento de linguagem natural e recuperação de informação para responder a perguntas em linguagem natural. A arquitetura do sistema envolve várias etapas, como análise de perguntas, recuperação de documentos, recuperação de frases e extração de respostas. Os resultados da participação na competição de QA da CLEF 2005 mostraram que o sistema teve um desempenho satisfatório, com uma precisão geral de 64,5%. O sistema foi particularmente

¹<https://www.linguateca.pt/>

²<https://www.linguateca.pt/CETEMPUBLICO/>

³<https://www.linguateca.pt/Floresta/principal.html>

⁴<https://www.linguateca.pt/CHAVE/>

⁵<https://www.publico.pt/>

eficaz em responder perguntas factuais, mas enfrentou desafios com perguntas de definição e perguntas temporalmente restritas. O artigo destaca a necessidade de melhorias no sistema, como o tratamento sintático de perguntas e respostas, resolução de anáforas e desambiguação semântica, o que poderia ser feito com a introdução de conhecimento semântico.

(QUARESMA; RODRIGUES) (2005) apresentam um sistema de QA desenvolvido para processar documentos jurídicos em português. Esse sistema foi aplicado a um conjunto de 180.000 documentos de várias instituições jurídicas portuguesas, incluindo tribunais superiores e o Ministério Público. O objetivo do sistema é facilitar o acesso a informações legais, melhorando a investigação criminal ao permitir que os investigadores encontrem processos criminais semelhantes e aprendam com erros passados, respondendo a perguntas sobre locais, datas, definições e casos específicos, além de indicar documentos relevantes. O processo inclui várias fases: pré-processamento dos documentos, análise das perguntas, seleção de documentos relevantes e inferência de respostas. As etapas incluem também a indexação para recuperação de informações, análise sintática em português, interpretação semântica e criação de uma ontologia específica. A avaliação do sistema foi realizada com 200 perguntas, resultando em 25% de respostas corretas. Em 46% dos casos, o sistema não forneceu resposta, principalmente devido à análise sintática incorreta, informações incompletas e desempenho insatisfatório do sistema de recuperação de informações. O artigo conclui que há espaço para melhorias significativas, especialmente no que se refere ao processamento semântico e sintático das perguntas e documentos.

O sistema SENSO foi proposto em (SAIAS; QUARESMA, 2007), onde os autores utilizaram um sistema modular composto de cinco módulos para procurar respostas para perguntas em português em uma lista de fatos. O primeiro passo consiste em importar uma coleção de textos contendo mais de 210 mil notícias e mais de 330 mil artigos da Wikipedia para o módulo de Biblioteca, onde os textos são indexados com o uso do LUCENE⁶. Cada texto é então processado pelo módulo de *Query* utilizando o analisador sintático PALAVRAS (BICK, 2000) para obter uma representação sintática do texto. O sistema utiliza o módulo de Ontologia contendo uma base de conhecimento inicial com informações semânticas que ajudam no processo de inferência. O módulo *Solver* utiliza uma ferramenta baseada em Lógica de Primeira Ordem para encontrar respostas diretas no texto. Por fim, o módulo de Interface Web exibe os resultados em uma interface gráfica amigável. O sistema foi avaliado com um conjunto de 200 perguntas, alcançando uma precisão geral de 42%. Os autores destacam a eficácia do sistema para perguntas

⁶<https://lucene.apache.org/>

do tipo “definição”, mas também observam desafios em outras categorias, como perguntas temporalmente restritas e listas.

Um sistema de QA tradicional de domínio geral chamado ESFINGE foi proposto em (COSTA, 2005). A solução utiliza informações redundantes extraídas da Web para encontrar as respostas. Os autores testaram três diferentes abordagens: procurar as respostas somente na coleção de documentos da tarefa de QA, nos documentos da tarefa e na web e somente na web. A solução que buscava procurar as respostas tanto nos documentos fornecidos para a tarefa quanto nos dados disponíveis na web apresentou melhores resultados.

Em (OLIVEIRA; COELHO; GOMES, 2014) é proposto um sistema que explora Bases de Conhecimento Lexicais (BCLs) em português, tais como PAPEL, CARTÃO, ONTO.PT, TEP e OPENWORDNET.PT, para responder perguntas automaticamente. O artigo introduz as BCLs mencionadas e seis algoritmos com o objetivo de responder a perguntas em português. O uso de algoritmos baseados em *Page Ranking* mostrou-se eficaz, respondendo corretamente 41% das 3890 perguntas, melhorando significativamente os resultados em comparação com as outras abordagens. Além disso, foi observado que BCLs maiores, criadas automaticamente, levaram aos melhores resultados, em contraste com as BCLs estruturadas em *synsets*.

O sistema IDSAY, proposto em (CARVALHO; DE MATOS; ROCIO, 2009) é um sistema de QA modular de domínio aberto para o português. O objetivo da solução é prover respostas curtas e precisas, através da busca em uma base de conhecimento que consiste de textos escritos em linguagem natural. A solução proposta realiza em uma primeira etapa a Análise da Pergunta, onde são extraídas algumas características sintáticas como Tokenização, Lematização e Stemização, para auxiliar a próxima etapa que é a Classificação do Tipo de Pergunta. Em um segundo momento, um módulo de Extração de Entidades extrai as entidades presentes no texto que servirão como base para um módulo de Geração de Consulta, que irá compor uma consulta que será submetida a uma coleção gigante de documentos indexados. A lista de documentos retornada pela execução da consulta passa por um módulo de Recuperação de Trechos e, por fim, pelo módulo de Validação das Respostas.

O sistema QAPTNET é proposto em (CARVALHO; SIMÕES; ALMEIDA, 2021) com o objetivo de resolver a tarefa de QA em português, utilizando *Deep Learning*. Usando o *Stanford Question Answering Dataset* (SQuAD) (RAJPURKAR et al., 2016), foram selecionados 400 artigos genéricos da Wikipedia junto com suas respectivas perguntas e

respostas para a criação de um *dataset* específico para o português. Embora a acurácia final resultante tenha sido de apenas 50%, os autores destacam a importância da criação do *dataset* disponibilizado, que pode suportar futuros desenvolvimentos na área de QA em português.

Como visto acima, vários trabalhos se propuseram a criar recursos para a língua portuguesa e sistemas para a resolução da tarefa de QA tradicional. Embora o foco deste trabalho seja na tarefa de KGQA, que utiliza grafos de conhecimento como base para a resposta de perguntas, é importante contrastar a grande quantidade de trabalhos que abordam a tarefa de QA tradicional em comparação com a escassez de trabalhos que tratam da tarefa de KGQA em português. Isto reforça ainda mais a importância de pesquisar novas soluções para a tarefa de KGQA para a língua portuguesa. Os dois sistemas encontrados que buscam realizar o desafio de KGQA para português são descritos abaixo.

3.2 KGQA em português

Os autores em (KETSMUR; RODRIGUES; TEIXEIRA, 2017) propõem um sistema de KGQA usando a DBpedia como KG e SPARQL como linguagem de consulta para responder a perguntas factuais em português. Para construir a consulta SPARQL, eles inicialmente identificam o tipo de pergunta (causal, lista ou definição). Em seguida, determinam as classes esperadas da DBpedia como respostas potenciais (pessoa, agente, lugar, jogo, etc). Depois, realizam uma análise morfofossintática da pergunta. O próximo passo é a Ligação de Entidade usando o sistema BABELNET (NAVIGLI; PONZETTO, 2012). O quinto passo consiste na Ligação de Relação, pesquisando todas as propriedades vinculadas às entidades identificadas nas etapas anteriores e comparando seus nomes com os *synsets* extraídos do BABELNET. Finalmente, a consulta SPARQL é construída, usando as entidades e relações vinculadas nas etapas anteriores. O sistema é avaliado em um conjunto de dados de 22 perguntas factuais geradas pelos autores. No entanto, apenas 15 tinham respostas correspondentes na DBpedia, das quais o sistema gerou uma resposta correta em 10 casos. Embora o resultado obtido seja promissor, vale ressaltar que os autores usaram um conjunto de dados com poucos exemplos, e este conjunto de dados não é publicamente acessível, com difícil reprodutibilidade e comparação. A classificação do tipo de pergunta é baseada em regras, mas os detalhes sobre a construção e o processo de classificação da consulta não são claros.

([SOUZA et al.](#)) (2020) propõem uma abordagem baseada em ontologia para responder perguntas feitas em português sobre fatos armazenados em um KG. Os autores primeiro executam a etapa de Ligação de Entidade comparando os termos da pergunta com os rótulos da ontologia. A segunda etapa é a Ligação de Relação, onde os nós da árvore sintática da pergunta são comparados com os nós indexados da ontologia. Após isso, as consultas SPARQL são construídas e classificadas. As respostas são apresentadas na forma de visualizações de dados, incluindo gráficos de barras, mostrando não apenas a resposta para a pergunta inicial, mas também outras respostas correlatas. Para avaliar o método, um conjunto de dados de perguntas em português foi construído com base em um subconjunto do QALD contendo 150 perguntas, adaptadas para o domínio de filmes e séries, associando as classes e indivíduos mencionados no dataset QALD com as classes e indivíduos no contexto do IMDb ⁷. Como resultado, os autores reportam um F1-score de 57%.

Embora as duas abordagens acima mencionadas proponham soluções para a tarefa de KGQA em português, elas construíram seus próprios conjuntos de dados para teste e não avaliaram seus métodos utilizando conjuntos de dados padrão de KGQA, como o QALD ([USBECK et al., 2017](#)) ou LCQuAD ([TRIVEDI et al., 2017](#)), o que dificulta a comparação com seus métodos. Abaixo destacamos algumas abordagens que, embora tenham sido aplicadas em inglês e não em português, trazem ideias inovadoras de soluções para a tarefa de KGQA.

3.3 KGQA em inglês

Em ([SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018](#)) é apresentado o FRANKENSTEIN, um *framework* de otimização de sistemas de KGQA que integra e seleciona componentes de KGQA para compor pipelines otimizados. O FRANKENSTEIN utiliza técnicas de aprendizado de máquina para selecionar componentes adequados com base nas características de uma pergunta, visando otimizar o F1-Score. O *framework* é composto por vários módulos, incluindo um extrator de características, classificadores de componentes de KGQA e um gerador de pipelines. Ele integra 29 componentes de KGQA reutilizáveis, que resolvem cinco subtarefas diferentes de KGQA, como Ligação de Entidade, Ligação de Relação e Construção da Consulta. O *framework* foi avaliado usando dois *benchmarks* renomados, QALD-5 e LCQuAD, mostrando a capacidade de combinar

⁷<https://www.imdb.com/>

componentes de QA para produzir pipelines otimizados que superam pipelines de QA estáticos. A avaliação de cada um dos 29 componentes revelou que o desempenho deles variou significativamente entre os conjuntos de dados, demonstrando que não há um componente de KGQA de melhor desempenho universal para todas as tarefas e perguntas. Como resultado, o FRANKENSTEIN atingiu um F1-score de 20% no QALD-5.

Algumas abordagens para KGQA utilizam métodos de *Neural Machine Translation* (NMT) para converter diretamente uma pergunta em linguagem natural em uma consulta SPARQL. Por exemplo, (RONY et al.) (2022) descreve o desenvolvimento do SGPT, um sistema inovador para gerar consultas SPARQL a partir de perguntas em linguagem natural. Este sistema aborda a complexidade de entender tanto a pergunta quanto os padrões do grafo de conhecimento subjacente. O SGPT codifica características linguísticas das perguntas e informações de sub-grafos e utiliza um modelo de linguagem generativo para produzir as consultas SPARQL. A performance do sistema foi avaliada usando conjuntos de dados públicos como LCQuAD, VQuAnDA (KACUPAJ et al., 2020) e QALD, alcançando um F1-score de 67.82%

(PURKAYASTHA et al.) (2022) propõem uma nova abordagem para o desafio de KGQA, apresentando uma arquitetura neural modular em duas etapas. A primeira etapa envolve um módulo de NMT que traduz uma pergunta em um esboço da consulta SPARQL desejada, chamada de ‘silhueta’ SPARQL. A segunda etapa inclui um Módulo de Pesquisa em Grafos Neurais, que visa melhorar a qualidade da silhueta SPARQL detectando as relações corretas no grafo de conhecimento subjacente. A abordagem é testada em dois conjuntos de dados KGQA diferentes: LCQuAD e QALD. O modelo *sequence-to-sequence* proposto alcançou um F1 de 83,08% no LCQuAD e 55,3% de Macro F1 no QALD-9.

Abordagens mais recentes também têm utilizado *Large Language Models* (LLMs) para a tarefa de KGQA, principalmente o *Generative Pretrained Transformer* (GPT). (TAN et al., 2023) investiga a capacidade do CHATGPT⁸ e outros modelos LLMs em substituir modelos tradicionais de KGQA. O estudo avaliou o ChatGPT e sua família de LLMs em oito conjuntos de dados de perguntas complexas baseadas em GC, incluindo seis em inglês e dois multilíngues, cobrindo aproximadamente 190.000 perguntas em 13 idiomas. No entanto, elas não garantem um desempenho melhor, já que o GPT-4 alcançou um F1-score de 57.2%, comparado a 46,19% para o GPT-3.5v3 e 38.54% para o GPT-3 no conjunto de dados QALD-9.

⁸<https://chat.openai.com/>

Embora as abordagens que utilizam modelos baseados em NMT sejam o estado da arte na tarefa de KGQA em inglês, treinar diretamente esses sistemas para uma língua com baixa disponibilidade de conjuntos de dados, como o português, é muito desafiador. Por conta disso, neste trabalho, utilizamos a estratégia de dividir a tarefa de conversão de uma pergunta em linguagem natural para uma consulta SPARQL em cinco subtarefas que são menos intensivas em dados. Entretanto, com o objetivo de tornar possível a reprodutibilidade e comparação do sistema que será desenvolvido, serão utilizados os *datasets* padrão QALD e LCQuAD, amplamente utilizados para *benchmark* da tarefa de KGQA.

4 KGQA_{PT}: Um sistema de KGQA para o Português

O sistema aqui proposto, denominado *Portuguese Knowledge Graph Question Answering* (KGQA_{PT}) é baseado em uma arquitetura modular inspirada no *framework* FRANKENSTEIN (SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018) e no sistema de geração de consultas SPARQL *Query Generator* (SQG) (CHEN; KOKAR; MOSKAL, 2021), seguindo o sistema modular proposto em (LIANG et al., 2021). Dadas as limitações de poucos recursos e ferramentas para o português, em vez de criar um sistema que faça uma conversão direta de uma pergunta em linguagem natural para uma consulta SPARQL, o KGQA divide a solução em componentes menores, que executam tarefas mais específicas. O KGQA_{PT} é composto de cinco componentes: Analisador Sintático, Classificador de Tipo de Pergunta, Mapeamento de Conceitos, Gerador de Consultas e Ranqueamento de Consultas. Na Figura 9 podemos visualizar como cada componente contribui para o processo de KGQA em português.

Em um primeiro passo, a pergunta de entrada, submetida em linguagem natural, é processada pelo Analisador Sintático a fim de extrair as seguintes características sintáticas do texto: *POS-tagging*, a Árvore de Dependência Sintática, *Lemma* e *Stemming*, que serão úteis para as etapas subsequentes. Em seguida, o Classificador do Tipo de Pergunta classifica a pergunta de entrada com base no tipo de resposta que ela requer: booleana, lista ou contagem. Essa classificação é muito importante para o componente Geração de Consultas funcione corretamente, pois é o tipo de pergunta que definirá o formato da consulta que deverá ser gerada.

O próximo passo consiste em fazer o mapeamento de trechos do texto da pergunta de entrada para entidades e relações existentes no Grafo de Conhecimento, tarefa essa realizada pelo módulo de Mapeamento de Conceitos. As entidades e relações identificadas nesse componente são passadas para o próximo módulo de Geração de Consultas, responsável por criar as consultas candidatas a responder à pergunta inicial. Todas as

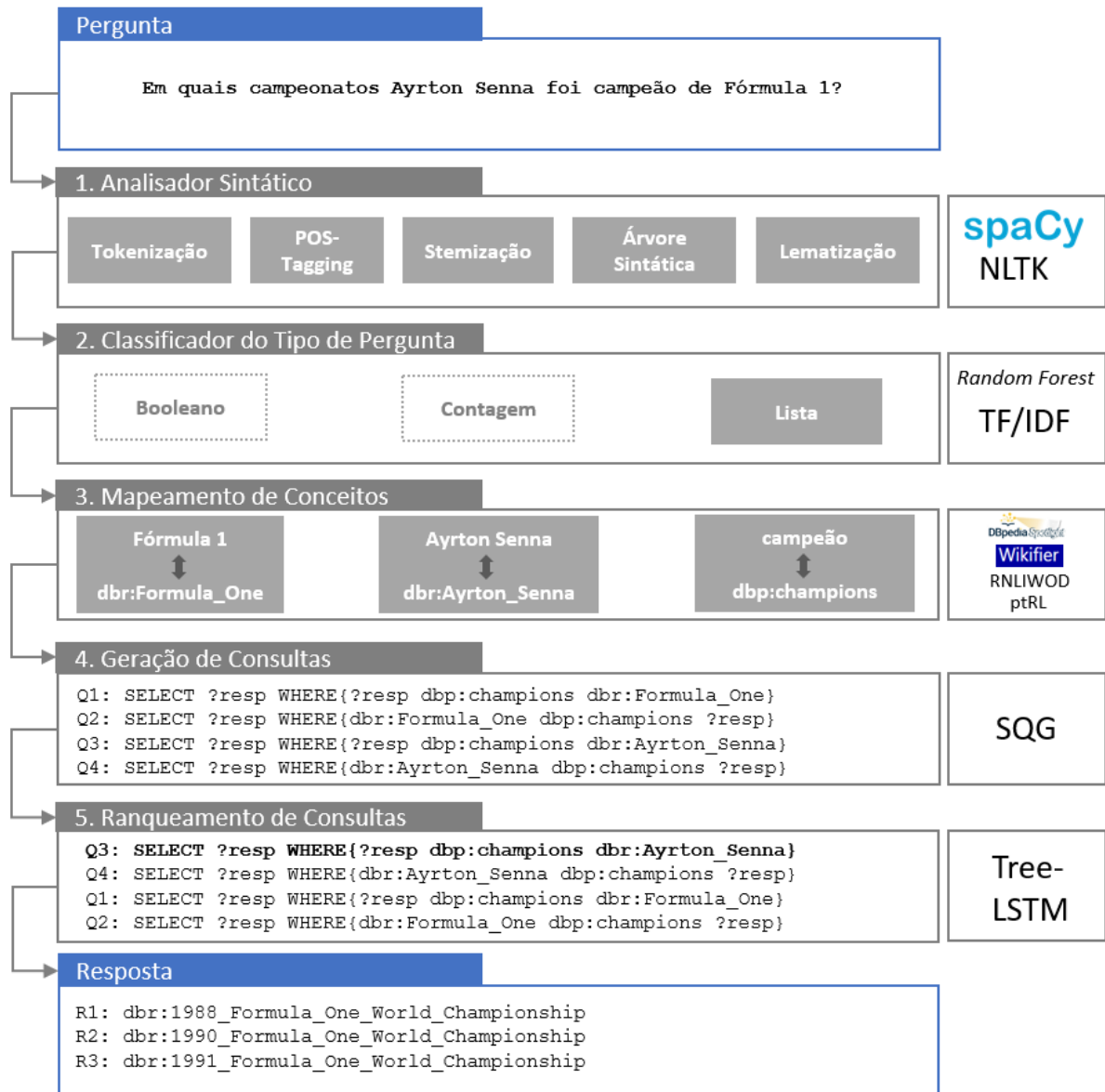


Figura 9: Sistema baseado em componentes para a tarefa de KGQA em português.

consultas geradas são passadas para o componente de Ranqueamento das Consultas que tem por objetivo ordenar as consultas de acordo com sua relevância. Para isso, é utilizado um modelo Tree-LSTM para avaliar a similaridade entre o texto da pergunta de entrada e a consulta gerada de forma que consultas mais similares à pergunta inicial são melhores ranqueadas. Por fim, a consulta melhor ranqueada é selecionada, executada no GC e a resposta é obtida. A seguir exploramos em mais detalhes cada um dos cinco componentes.

4.1 Analisador Sintático

O componente Analisador Sintático tem como objetivo extrair várias características sintáticas da pergunta de entrada, que ajudarão na execução dos componentes seguintes, tais como Tokenização, POS-Tagging, Lematização, Stemização e Árvore de Dependência Sintática. Na Figura 10 podemos ver um exemplo das características sintáticas com base na frase: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*”

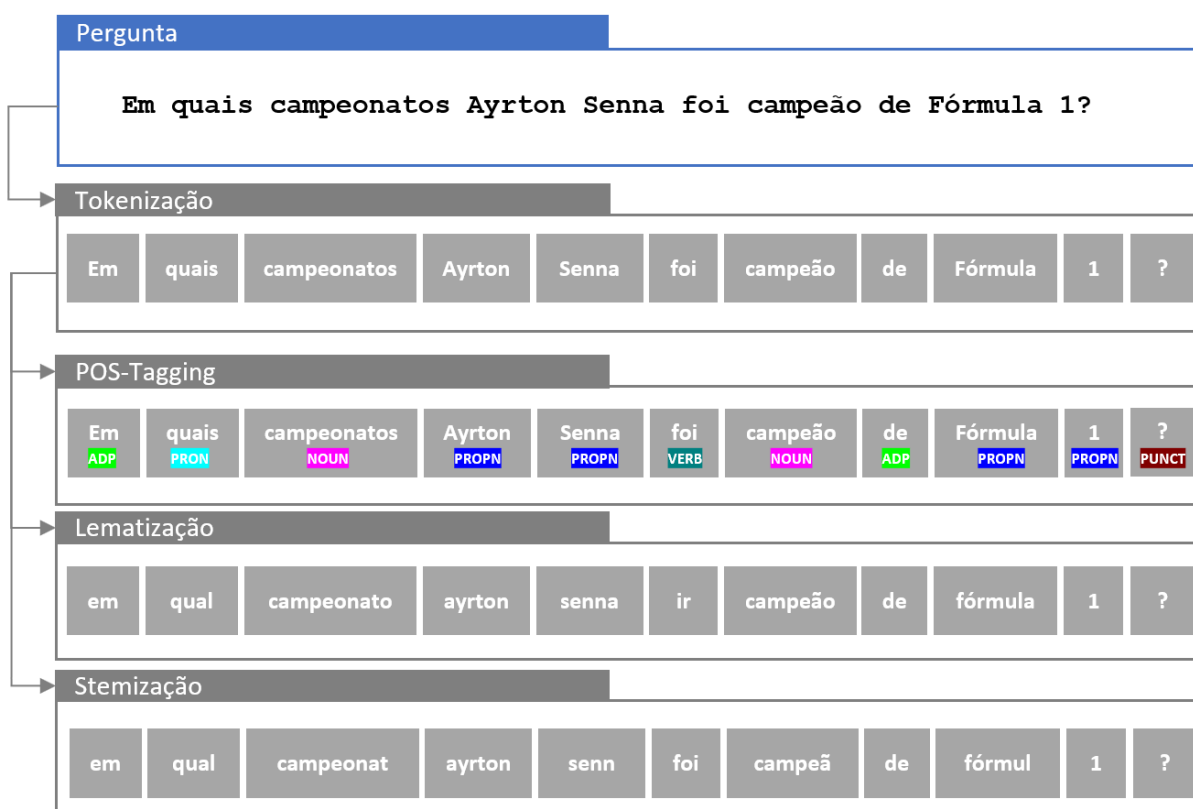


Figura 10: Características sintáticas referentes à frase: *Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*

O primeiro passo é a Tokenização da pergunta de entrada. Para isso, usamos a biblioteca SPACY¹ (HONNIBAL et al., 2020) com o modelo PT_CORE_NEWS_SM². **Tokenização** é um processo que consiste em dividir uma sentença em unidades menores de texto (STENGEL, 2007) que podem ser palavras, subpalavras ou até mesmo letras, permitindo assim, quebrarmos um texto grande em unidades menores de significado. A forma mais comum de Tokenização é a nível de palavra, que é exatamente a que é aplicada no módulo de Analisador Sintático. Tomando como exemplo a frase “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*”, o tokenizador devolve a seguinte lista de palavras: “Em”, “quais”, “campeonatos”, “Ayrton”, “Senna”, “foi”, “campeão”, “de”, “Fórmula”, “1” e “?”.

¹<https://spacy.io/>

²<https://spacy.io/models/pt>

Em seguida, cada *token* é anotado com seu POS-*Tag*, também obtido com SPACY. **Part-Of-Speech Tagging (POS-Tagging)** consiste em atribuir a cada palavra (*token*) a sua função sintática dentro do texto (THANAKI, 2017), em outras palavras, identificar a parte do discurso (*part-of-speech*) de cada palavra. Em termos práticos, o objetivo é classificar cada palavra como um substantivo, verbo, pronome, preposição, artigo, pontuação, etc. Na frase de exemplo, a anotação de POS-*Tag* ficaria assim: “Em = ADP”, “quais = PRON”, “campeonatos = NOUN”, “Ayrton = PROPN”, “Senna = PROPN”, “foi = VERB”, “campeão = NOUN”, “de = ADP”, “Fórmula = PROPN”, “1 = PROPN”, “?” = PUNCT”.

Para a Lematização, usamos o sistema SIMPLEMMA³ (BARBARESI, 2023), por demonstrar uma boa precisão em português. A **Lematização** é uma técnica que consiste em reduzir um conjunto de palavras por sua palavra base, diminuindo assim significativamente a quantidade de palavras processadas (GOYAL; PANDEY; JAIN, 2018). Por exemplo, as palavras “*estudar*”, “*estudaria*”, “*estudarão*”, “*estudaríamos*” e “*estudávamos*” são todas lematizadas para a palavra “*estudar*”. Vale ressaltar que o resultado da lematização sempre é uma palavra que realmente existe na gramática. Para o nosso exemplo, o processo de Lematização devolveria o seguinte: “em”, “qual”, “campeonato”, “ayrton”, “senna”, “ir”, “campeão”, “de”, “fórmula”, “1” e “?”.

Para o processo de Stemização, usamos o RSLPSTEMMER da biblioteca NLTK⁴ (BIRD; KLEIN; LOPER, 2009). A **Stemização** é um processo muito parecido com a Lematização, a diferença é que não há garantia de que o termo resultante seja uma palavra existente na gramática (HAGIWARA, 2021). Ou seja, a stemização simplesmente ‘corta’ as palavras para o seu radical comum sem se preocupar se o termo resultante existe realmente. Para o nosso exemplo, a Stemização devolveria o seguinte: “em”, “qual”, “campeonat”, “ayrton”, “senn”, “foi”, “campeã”, “de”, “fórmul”, “1” e “?”. Note que as palavras “campeonat” e “fórmul” não são palavras gramaticalmente corretas.

Por fim, para a extração da Árvore de Dependência Sintática referente à pergunta de entrada, também usamos a biblioteca SPACY. A **Árvore de Dependência Sintática** consiste no processo de analisar a estrutura sintática de uma sentença e estabelecer as relações estruturais de dependência entre as palavras que compõe o texto analisado. Dessa forma, cada palavra é classificada em um nível da árvore, partindo da palavra raiz. Na Figura 11 podemos visualizar a árvore que representa a pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*”

³<https://github.com/adbar/simplemma>

⁴<https://github.com/nltk/nltk>

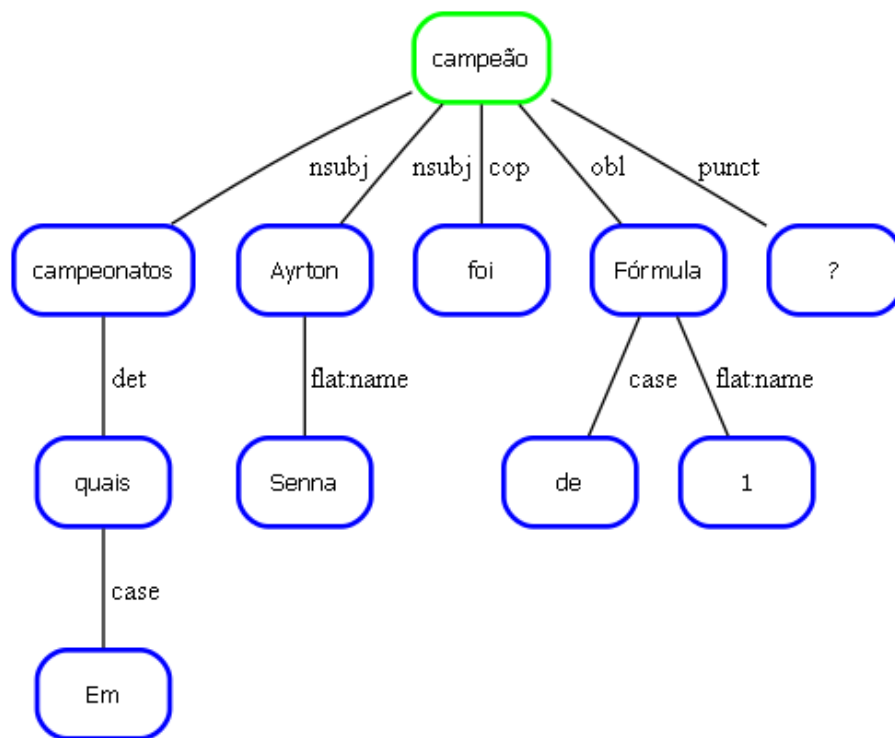


Figura 11: Árvore de Dependência Sintática referente à pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*”

Todos os passos realizados pelo módulo Analisador Sintático servem como entradas para os componentes subsequentes. Especificamente, a Árvore de Dependência Sintática e a representação POS-Tag serão entradas para o componente de Classificação de Consultas. As representações obtidas por meio da Lematização são empregadas no treinamento do Classificador do Tipo de Pergunta, enquanto a Stemização no componente de Mapeamento de Conceitos.

4.2 Classificador do Tipo de Pergunta

Para uma correta construção da consulta na linguagem SPARQL é necessário identificar antecipadamente qual o tipo de pergunta e, conseqüentemente, o tipo de resposta requerida. Por exemplo, existem perguntas que exigem respostas do tipo ‘Sim’ ou ‘Não’ que chamamos de *booleanas*, como na pergunta: “*Ayrton Senna foi um piloto de Fórmula 1?*”. No Exemplo 4.1, observamos que a consulta SPARQL que responde corretamente a essa pergunta inicia com a cláusula **ASK**.

```
ASK {  
  dbr:Ayrton_Senna rdf:type dbo:FormulaOneRacer  
}  
# Output:  
# true
```

Exemplo 4.1: Query SPARQL correspondente à pergunta: “*Ayrton Senna foi um piloto de Formula 1?*”

Existem também aquelas perguntas cuja resposta é na verdade uma lista de valores, como no caso da pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1 ?*”. Nesse caso, note que a estrutura da consulta SPARQL é formada com o uso das cláusulas `SELECT` e `WHERE`, conforme podemos observar no Exemplo 4.2.

```
SELECT ?resp WHERE {  
  ?resp dbp:champions dbr:Ayrton_Senna  
}  
# Output:  
# http://dbpedia.org/resource/1988_Formula_One_World_Championship  
# http://dbpedia.org/resource/1990_Formula_One_World_Championship  
# http://dbpedia.org/resource/1991_Formula_One_World_Championship
```

Exemplo 4.2: Query SPARQL correspondente à pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Formula 1?*”

Por fim, as perguntas também podem ser do tipo cuja resposta exige apenas um número, como na questão: “*Quantas vezes Ayrton Senna foi campeão da Fórmula 1?*”. Como vemos no Exemplo 4.3, a consulta SPARQL é muito parecida com a do Exemplo 4.2, com a diferença que foi necessário inserir a cláusula `COUNT`.

```
SELECT COUNT(?resp) WHERE {  
  ?resp dbp:champions dbr:Ayrton_Senna  
}  
# Output:  
# 3
```

Exemplo 4.3: Query SPARQL correspondente à pergunta: “*Quantas vezes Ayrton Senna foi campeão da Fórmula 1?*”

Como visto, uma consulta SPARQL pode ter uma estrutura diferente dependendo do tipo da pergunta. Sendo assim, podemos concluir que a etapa de classificação do tipo de pergunta é muito importante para o sucesso do modelo de perguntas e respostas, pois tal classificação será crucial para definir se o componente de Geração das Consultas irá ou não produzir uma consulta correta. Para a construção do Classificador do Tipo de Pergunta, foi utilizado o *dataset* LCQuAD-1 que contém 5000 perguntas, com suas respectivas consultas SPARQL que respondem à pergunta. Como o LCQuAD-1 estava disponível somente em inglês, as perguntas foram traduzidas automaticamente para o português utilizando a API do Google Translate ⁵. Cada pergunta foi anotada como pertencente a uma das três possíveis classes: Booleano, Contagem ou Lista. Para o processo de anotação, utilizou-se a seguinte regra: Caso a consulta contivesse as cláusulas SELECT e COUNT o exemplo era anotado como ‘Contagem’, caso tivesse a cláusula ASK, classificado como ‘Booleano’ e, caso contrário, a anotação atribuída era de ‘Lista’. Após isso, o *dataset* foi dividido em 80% para treino e 20% para teste.

Foram testados três algoritmos de classificação: *Random Forest* (RF), *Support Vector Machine* (SVM) e *Neural Network* (NN) e dois tipos de entradas: vetor de TF/IDF e *embedding*. Inicialmente, utilizamos a representação lematizada do texto da pergunta de entrada que foi produzido pelo módulo de Analisador Sintático citado na seção 4.1 para criar um vetor de TF/IDF que serviu de entrada para o treinamento dos modelos. Uma outra rodada de treinamento foi realizada, substituindo o vetor de TF/IDF pelo *embedding* médio do texto lematizado, usando os *embeddings* do FASTTEXT (JOULIN et al., 2017). Cada modelo teve seus parâmetros otimizados utilizando *Grid Search* e usando 10 *folds* para validação cruzada. Na avaliação dos resultados dos modelos, que será explicada em detalhes no Capítulo 5, o modelo RF com TF/IDF obteve o melhor resultado de classificação, registrando um F1-macro de 99.4% e, por isso, foi escolhido como o Classificador do Tipo de Pergunta.

4.3 Mapeamento de Conceito

O componente de **Mapeamento de Conceito** consiste em relacionar conceitos (entidades ou relações) presentes no texto com os recursos, classes e propriedades do Grafo de Conhecimento. Para que tal relacionamento funcione de maneira correta, duas subtarefas são de extrema importância: a Ligação de Entidade e a Ligação de Relação. O GC utilizado foi a DBpedia, que contém três possíveis tipos de entidades: Recursos, Classes

⁵<https://cloud.google.com/translate>

e Propriedades. Os conceitos identificados no texto que representam entidades como pessoas, lugares e coisas foram mapeados para Recursos ou Classes da DBpedia. Por sua vez, os conceitos extraídos que simbolizam relações entre duas entidades, foram mapeados para as Propriedades da DBpedia.

Na DBpedia, um **Recurso** é uma entidade concreta ou abstrata e pode ser representada por um identificador *Internationalized Resource Identifier* (IRI) ou por um rótulo textual. Por exemplo, na pergunta “*Onde nasceu Ayrton Senna?*”, a entidade “*Ayrton Senna*” é mapeada para o recurso da DBpedia correspondente que possui o seguinte IRI: “https://dbpedia.org/resource/Ayrton_Senna”. Já uma **Classe** é um tipo especial de Recurso que é utilizado para agrupar recursos em um nível maior de abstração. Por exemplo, o recurso “*Rio de Janeiro*”, identificado pelo IRI: “https://dbpedia.org/resource/Rio_de_Janeiro” pertence à classe “*City*”, cujo IRI é: “<https://dbpedia.org/resource/City>”.

Por fim, uma **Propriedade** também é um tipo especial de recurso na DBpedia, mas é utilizada para representar relacionamentos entre outros recursos ou apenas para descrever atributos de um recurso em particular. Por exemplo, a propriedade “*dbo:fastestDriver*”, cujo IRI é: <http://dbpedia.org/ontology/fastestDriver> é utilizada para relacionar os recursos “*dbr:/Ayrton_Senna*”, representando o piloto de Fórmula 1, e “*dbr:1993-European_Grand_Prix*”, representando a competição a qual ele participou e foi eleito o piloto mais rápido.

Em inglês, existem vários sistemas que realizam LE com bons resultados. Podemos citar por exemplo: SPEL (SHAVARANI; SARKAR, 2023), ENTQA (ZHANG; HUA; STRATOS, 2022), REFINED (AYOOLA et al., 2022) e CHOLAN (RAVI et al., 2021), dentre vários outros. Já para o português, foram encontrados apenas dois sistemas que realizam a tarefa de LE: DBPEDIA SPOTLIGHT (DAIBER et al., 2013; MENDES et al., 2011) e WIKIFIER (BRANK; LEBAN; GROBELNIK, 2017).

DBpedia Spotlight (MENDES et al., 2011) é uma solução de LE que automaticamente mapeia e anota menções de recursos da DBpedia em um texto de entrada fornecido em linguagem natural. Primeiramente são detectados trechos do texto de entrada que possam estar ligados a recursos da DBpedia. Depois, é realizado um processo de desambiguação das entidades detectadas através de um modelo generativo probabilístico. Por fim, é feito um processo de indexação das entidades extraídas para mapear de forma mais eficiente as entidades correspondentes da DBpedia. O sistema está disponível em 16 línguas, incluindo inglês, espanhol e português.

Wikifier (BRANK; LEBAN; GROBELNIK, 2017) é uma ferramenta de LE que utiliza um método baseado em *Pagerank* para identificar conceitos relevantes em um texto de entrada submetido em linguagem natural e mapeá-los para a DBpedia, fortemente baseado nos dados de páginas da Wikipedia. Inicialmente são identificados palavras ou frases que se referem a um conceito da Wikipedia. Em seguida, determina-se qual é exatamente o recurso da DBpedia ao qual o conceito extraído do texto se refere. Por fim, são definidos quais recursos da DBpedia são relevantes, levando em consideração todo o texto de entrada, e que devam ser incluídos como uma resposta de saída do sistema. O modelo está disponível atualmente em 134 línguas, incluindo o português.

Para realizar a tarefa de LE para o português, o texto da pergunta de entrada foi submetido para ambos os sistemas DBpedia Spotlight ⁶ e Wikifier ⁷ e as respostas foram combinadas através da união dos resultados dos dois modelos a fim de gerar um resultado mais consistente, uma vez que a combinação de resultados minimiza as fraquezas individuais de cada sistema, mas ao mesmo tempo maximiza suas forças, produzindo os melhores resultados possíveis (SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018).

O mapeamento de conceitos presentes no texto de entrada que representam relações e as suas respectivas propriedades no GC é conhecido como Ligação de Relação (LR). Os modelos para LR são muito mais escassos que os de EL, mesmo para o inglês (SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018). Em inglês, os seguintes sistemas estão disponíveis para a tarefa de LR: EARL (DUBEY et al., 2018), FALCON (SAKOR; SINGH; VIDAL, 2019) e RNLIWOD (SINGH; RADHAKRISHNA, A S; BOTH; SHEKARPOUR; LYTRA; USBECK; VYAS; KHIKMATULLAEV; PUNJANI; LANGE et al., 2018). Destes, apenas o RNLIWOD pôde ser utilizado para o português, através da adaptação de alguns de seus componentes para lidar com as especificidades do português, como adaptação do *Stemmer* e tradução do dicionário de relações.

O sistema RNLIWOD faz o mapeamento entre as relações identificadas em um texto e as suas respectivas propriedades na DBpedia com base em um dicionário de propriedades e classes junto com as suas respectivas descrições. O dicionário é composto por um conjunto de triplas do tipo (sujeito, predicado, objeto), onde o sujeito representa o IRI da relação dentro do GC, o predicado é a propriedade `rdfs:label` e o objeto é o rótulo da propriedade. Para adaptar o modelo para o português, todos os rótulos das

⁶<https://api.dbpedia-spotlight.org/pt/annotate>

⁷<http://www.wikifier.org/annotate-article>

propriedades do dicionário foram traduzidos do inglês para o português utilizando a API Google Translate. No Exemplo 4.4 podemos observar a estrutura do arquivo contendo um exemplo de três triplas que estão contidas no arquivo de dicionário de propriedades original e sua versão depois da tradução.

Inicialmente, o RNLIWOD obtém a versão stemizada da pergunta de entrada que foi processada pelo módulo Analisador Sintático. Depois, os rótulos das propriedades do dicionário também passam pelo processo de stemização. Por fim, é feita uma busca pelo rótulo stemizado da propriedade no texto stemizado da pergunta de entrada. Uma vez que um rótulo de propriedade é encontrado no texto da pergunta, admite-se que aquela parte do texto faz referência à propriedade da DBpedia mapeada. Na Figura 4.3 podemos observar um esquema mostrando um exemplo de mapeamento de relações para a pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Formula 1?*” usando o RNLIWOD adaptado para o português.

```
#Antes:
dbo:seatingCapacity rdfs:label "capacity" .
dbp:pastMembers rdfs:label "past members" .
dbo:order rdfs:label "belong to" .

#Depois:
dbo:seatingCapacity rdfs:label "capacidade" .
dbp:pastMembers rdfs:label "Membros antigos" .
dbo:order rdfs:label "pertence a" .
```

Exemplo 4.4: Exemplo do dicionário de propriedades da DBpedia junto com seus textos descritivos traduzidos para o português.

Como percebido, o sistema de mapeamento RNLIWOD é muito simples e consiste apenas da busca de um texto descritivo da propriedade da DBpedia na pergunta de entrada e, por isso, não consegue lidar com pequenas variações dos rótulos. Por conta disso, propomos o PTRL, uma evolução do RNLIWOD que utiliza os vetores de *embeddings*, produzidos pelo modelo FASTTEXT (BOJANOWSKI et al., 2017) dos rótulos das propriedades, para identificar as relações presentes no texto da pergunta. O método PTRL primeiramente identifica e remove do texto de entrada todas as entidades identificadas pelos modelos de LE, além das *stop words*, que são palavras acessórias do texto que não



Figura 12: Esquema de funcionamento do método de Ligação de Relações RNLIWOD adaptado para o português

trazem valor semântico, tais como conjunções, preposições e artigos. Por último, são removidos os pronomes interrogativos, como por exemplo, “onde”, “quem”, “quando”, etc.

A hipótese do PTRL é que removendo as entidades, as *stop words* e os pronomes interrogativos do texto da pergunta de entrada, restará somente o texto referente à relação existente. Por exemplo, na pergunta “Onde nasceu Ayrton Senna?”, removendo o texto referente à entidade “Ayrton Senna” que foi ligada ao GC pelos modelos de LE, restará o texto “Onde nasceu?”. Removendo agora as *stop words* e os pronomes interrogativos, restará apenas “nasceu”, que é exatamente a relação que precisamos mapear no GC. De posse do texto remanescente da pergunta inicial, o PTRL utiliza o modelo pré-treinado para o português do FASTTEXT (BOJANOWSKI et al., 2017) para calcular o vetor de *embedding* do texto que representa a relação que se quer ligar ao KG. No nosso exemplo,

iremos obter o *embedding* da palavra “*nasceu*”. Esse vetor de *embedding* é comparado, via similaridade de cosseno, com todos os outros vetores que foram calculados previamente com base nos rótulos das propriedades do dicionário de propriedades. As propriedades com os 3 maiores valores para a similaridade de cosseno são selecionadas e mapeadas como possíveis relações candidatas. A Figura 13 ilustra as etapas envolvidas no modelo PTRL.

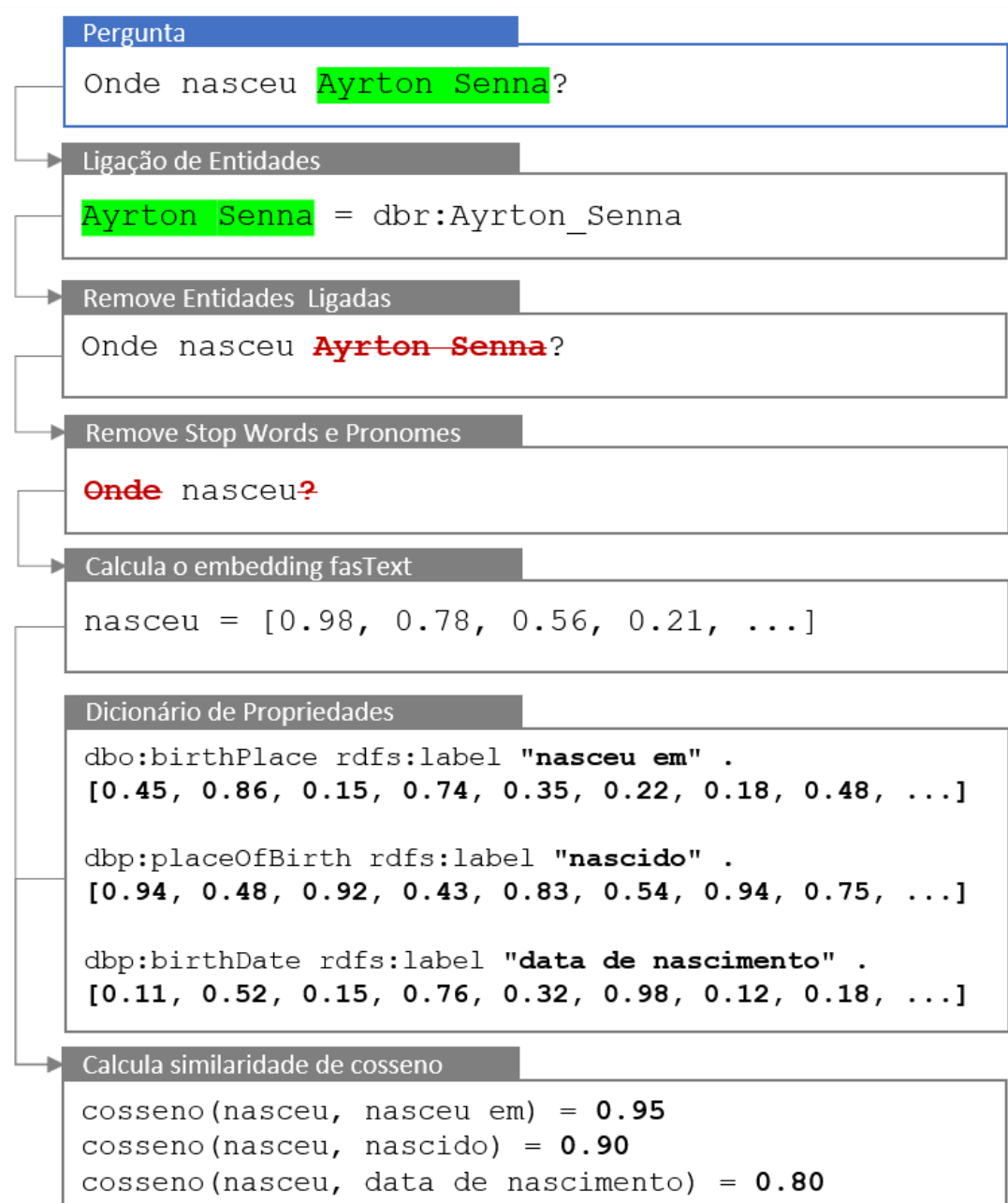


Figura 13: Resumo do funcionamento do **ptRL** para Ligação de Relações.

Embora o PTRL não tenha sido testado em um *dataset* de LE, por conta da indisponibilidade deste tipo de recurso em português, pretendemos avaliar seu desempenho executando o KGQA_{PT} com e sem o PTRL e comparar os resultados. No sistema aqui

desenvolvido, para realizar a tarefa de Ligação de Relação para o português, o texto da pergunta de entrada foi submetido para ambos os sistemas RNLIWOD adaptado para o português e o PTRL. Os resultados dos dois modelos foram combinados pela união com o objetivo de melhorar o resultado final.

4.4 Geração de Consultas

O componente de Geração de Consultas é responsável por construir as consultas SPARQL com base em três informações dos componentes anteriores: o tipo de pergunta, as entidades mapeadas e as relações identificadas. O componente de Classificação do Tipo de Pergunta, abordado na seção 4.2, é crucial para um bom resultado do modelo como um todo, visto que é ele que definirá o formato da primeira parte da consulta que será gerada pelo componente de Geração das Consultas. Ou seja, a primeira parte da consulta SPARQL será definida como `ASK`, `SELECT` ou `SELECT COUNT()` a depender do resultado da classificação do módulo de Classificação do Tipo de Pergunta. A segunda parte da consulta será responsável pela geração da cláusula `WHERE` da consulta SPARQL e consiste basicamente na criação de uma ou mais triplas no formato (sujeito, predicado, objeto). Portanto, o objetivo principal do módulo de Geração das Consultas é gerar uma lista de triplas para compor a cláusula `WHERE` da consulta.

Para o processo de construção das consultas foi utilizado o SQG, método proposto em (ZAFAR; NAPOLITANO; LEHMANN, 2018). O método consiste basicamente em utilizar as listas de entidades e de relações mapeadas do componente de Mapeamento de Conceitos, para gerar como saída uma lista de triplas que formarão a parte `WHERE` da consulta SPARQL. A hipótese do SQG é que a representação formal da pergunta de entrada é dada por um caminho no GC que contenha somente o conjunto de entidades mapeadas E , o conjunto de relações mapeadas R e também os nós de resposta. Inicialmente são identificados caminhos válidos no GC a partir das entidades $e \in E$ identificadas no texto da pergunta e percorrendo o GC. No entanto, por conta do tamanho gigantesco dos Grafos de Conhecimento (GCs) existentes, seria computacionalmente muito custoso mapear todos os caminhos válidos. A estratégia para contornar esse problema é identificar um subgrafo que contenha somente as entidades e relações identificadas no texto da pergunta de entrada. Ou seja, utilizamos somente as entidades $e \in E$ e as relações $r \in R$ e passamos a procurar caminhos válidos com base nesse subgrafo em vez de em todo o Grafo de Conhecimento.

Para a construção do subgrafo inicial, primeiro são criados os nós que representam as entidades E , conforme podemos ver na linha 3 do Algoritmo 1. Em seguida, são inseridas as relações mapeadas pelo componente de Mapeamento de Conceito, somente se tais relações realmente estejam conectadas a alguma das entidades $e \in E$ já inseridas no subgrafo. Caso a relação exista no GC, criamos uma entidade virtual v e conectamos a relação mapeada r à entidade mapeada e e à entidade virtual recém criada v , considerando os dois sentidos da relação: (e, r, v) e (v, r, e) , linhas 4 a 11. O objetivo de considerarmos os dois sentidos da relação é enriquecer o subgrafo e aumentar a chance de mapearmos corretamente a relação identificada. Uma vez concluído esse processo, todas as relações que estejam conectadas diretamente a uma entidade mapeada serão inseridas, ou seja, estamos mapeando possíveis nós resposta (entidades virtuais v) que estejam a uma distância de um salto no GC das entidades e .

No entanto, a depender da complexidade da pergunta, a resposta correspondente pode estar localizada a mais de um salto de distância das entidades e mapeadas, por isso, somente mapear entidades ligadas diretamente àquelas identificadas no texto inicial da pergunta pode não ser suficiente para responder corretamente à questão. Para atender a essa necessidade, o subgrafo é expandido a partir das entidades inseridas na etapa anterior, ou seja, as entidades $e \in E$ e as virtuais v . Para essa expansão, utilizamos as mesmas relações mapeadas r , conectando-as a novas entidades virtuais criadas v' , considerando ambos os sentidos das relações, ou seja, serão adicionadas ao subgrafo as novas triplas (e, r, v') ou (v, r, v') e também as triplas (v', r, e) ou (v', r, v) , conforme podemos observar nas linhas 12 a 27 do Algoritmo 1. O objetivo dessa expansão é tentar capturar respostas às perguntas mais complexas, ou seja, que possuam fatos complexos e que não possam ser respondidas apenas mapeando fatos simples. É claro que esse processo poderia continuar indefinidamente, através da expansão do subgrafo G utilizando as entidades presentes nele e as relações identificadas no texto da pergunta. Mas a cada etapa, mais e mais entidades virtuais seriam criadas e isso deixaria o subgrafo cada vez maior e inviabilizaria computacionalmente a geração das consultas. Por isso, restringimos esse processo para apenas dois saltos e enumeramos todos os possíveis caminhos no subgrafo G para gerar uma lista de caminhos candidatos.

De posse do subgrafo G que contém as entidades $e \in E$ e as relações $r \in R$ identificadas no texto inicial da pergunta, além das novas entidades virtuais v e v' adicionadas, precisamos extrair desse subgrafo os possíveis caminhos que respondem à pergunta. Para isso, consideramos todas as entidades virtuais v e v' como possíveis nós de resposta, mas somente se fazem parte de um caminho válido no grafo. Um caminho W é considerado

Algoritmo 1 Construção do subgrafo

```

1: Dados:  $E, R, K$ 
2: Inicializar  $G$  como um grafo vazio;
3:  $\forall e \in E$  adicionar  $e$  como um nó de  $G$ ;
4: para cada  $e \in E, r \in R$  faça
5:   se  $(e, r, ?) \in K$  então
6:     Inserir  $(e, r, ?)$  em  $G$ ;
7:   fim se
8:   se  $(?, r, e) \in K$  então
9:     Inserir  $(?, r, e)$  em  $G$ ;
10:  fim se
11: fim para
12: para cada  $(e_1, r, e_2) \in G$  faça
13:   para cada  $r' \in R, r' \neq r$  faça
14:     se  $(e_1, r', ?) \in K$  então
15:       Inserir  $(e_1, r', ?)$  em  $G$ ;
16:     fim se
17:     se  $(?, r', e_1) \in K$  então
18:       Inserir  $(?, r', e_1)$  em  $G$ ;
19:     fim se
20:     se  $(e_2, r', ?) \in K$  então
21:       Inserir  $(e_2, r', ?)$  em  $G$ ;
22:     fim se
23:     se  $(?, r', e_2) \in K$  então
24:       Inserir  $(?, r', e_2)$  em  $G$ ;
25:     fim se
26:   fim para
27: fim para

```

válido se todas as entidades $e \in W$ e relações $r \in W$ identificadas no texto da pergunta pertencem a esse caminho. No caso em que há somente um caminho válido identificado, esse será retornado como aquele que irá gerar a consulta que responde corretamente à pergunta inicial. Caso seja identificado mais de um caminho válido que possivelmente responda à pergunta, tais caminhos irão gerar várias consultas que precisarão ser ranqueadas posteriormente de acordo com a sua similaridade com o texto da pergunta de entrada.

A identificação do tipo da pergunta é de vital importância para o sistema, pois é esse componente que definirá a estrutura da consulta SPARQL que será criada. Após a extração dos caminhos candidatos, o tipo de pergunta é classificado utilizando o componente de Classificação do Tipo de Pergunta e a consulta é construída de acordo com o tipo identificado. Para perguntas do tipo booleano, a cláusula **ASK** é utilizada. Já para as perguntas que exigem uma lista de valores, a cláusula **SELECT** é usada. Por fim, para os casos em que a pergunta exige uma contagem de valores, as cláusulas **SELECT COUNT()** são utilizadas.

Em alguns casos, uma única relação identificada no texto da pergunta inicial deve ser utilizada mais de uma vez no processo de geração da consulta para que a pergunta seja respondida corretamente. Por exemplo, considere a pergunta “*Quem namorou Pelé e Ayrton Senna?*” Fica claro nesse caso que a relação “*namorou*” (“**dbo:partner**” na DBpedia) deve estar ligada às duas entidades: “*Pelé*” (“**dbr:Pe1é**” na DBpedia) e “*Ayrton Senna*” (“**dbr:Ayrton_Senna**” na DBpedia). Ou seja, no processo de geração de triplas, devemos duplicar a relação “**dbo:partner**” para conseguirmos responder com sucesso à pergunta. O Exemplo 4.5 ilustra como ficaria a consulta correta, note que foi necessário repetir a relação “**dbo:partner**”.

Para resolver esse problema, foi desenvolvido um Classificador de Relações Duplas para o qual foram testados três algoritmos: *Random Forest* (RF), *Support Vector Machine* (SVM) e *Neural Network* (NN) e dois tipos de entrada: TF/IDF e vetor de *embedding* do texto. Utilizamos inicialmente a representação lematizada do texto da pergunta que foi produzida pelo módulo de Analisador Sintático citado na seção 4.1 para criar um vetor de *embedding* ou de TF/IDF do texto. O vetor calculado é submetido aos modelos selecionados que classificam a pergunta como contendo ou não relações duplas. Caso a pergunta seja classificada como possuindo relações duplicadas, uma nova tripla é adicionada à lista de triplas já criada. O conjunto de dados utilizado para treinamento do Classificador de Relações Duplas foi o LCQuAD, visto que se trata de um *dataset* que possui perguntas

mais complexas com muitos casos de relações duplas. Dos modelos testados, o modelo de *Neural Network* com *embedding* obteve o melhor resultado, alcançando um F1-macro de 73.7% e, por isso, foi escolhido como o Classificador de Relações Duplas.

Nesse ponto, as consultas SPARQL precisam ser ranqueadas de forma que a melhor consulta seja selecionada para responder à pergunta.

```
SELECT ?resp WHERE {  
  ?resp dbo:partner dbr:Ayrton_Senna .  
  ?resp dbo:partner dbr:Pelé .  
}  
# Output:  
# http://dbpedia.org/resource/Xuxa
```

Exemplo 4.5: Consulta SPARQL correspondente à pergunta: “*Quem namorou Pelé e Ayrton Senna?*”.

4.5 Ranqueamento das Consultas

Uma vez que o módulo de Geração das Consultas tenha gerado uma lista de possíveis consultas que respondam à pergunta, precisamos ordená-las, segundo algum critério, de forma que a consulta com a maior probabilidade de responder corretamente à pergunta seja selecionada. O método de ranqueamento adotado é o proposto em (LIANG et al., 2021), onde a hipótese principal é que as consultas com maior probabilidade de responderem corretamente à pergunta, são aquelas cuja sua representação em árvore seja mais similar à árvore de dependência sintática do texto de entrada. Tal similaridade é calculada através de um modelo Tree-LSTM (TAI; SOCHER; MANNING, 2015), no qual a árvore que representa a pergunta de entrada é comparada com a árvore obtida da consulta gerada. Um modelo de Tree-LSTM é muito parecido com o tradicional modelo LSTM *Long Short Term Memory*, com a diferença que leva em conta também a estrutura em árvore das palavras de um texto e não somente a sua sequência na frase como faz o LSTM. Na Figura 14 podemos ver um diagrama comparativo entre a arquitetura de uma LSTM tradicional e uma Tree-LSTM.

O modelo Tree-LSTM é uma extensão natural das arquiteturas LSTM, mas permitindo a propagação de informações hierárquicas e não somente sequenciais. Uma unidade Tree-LSTM difere de uma unidade LSTM padrão por ter vetores de portão e atualizações

de células de memória dependentes dos estados de suas unidades filhas. Além disso, em vez de um único portão de esquecimento, a unidade Tree-LSTM contém um portão de esquecimento para cada filha.

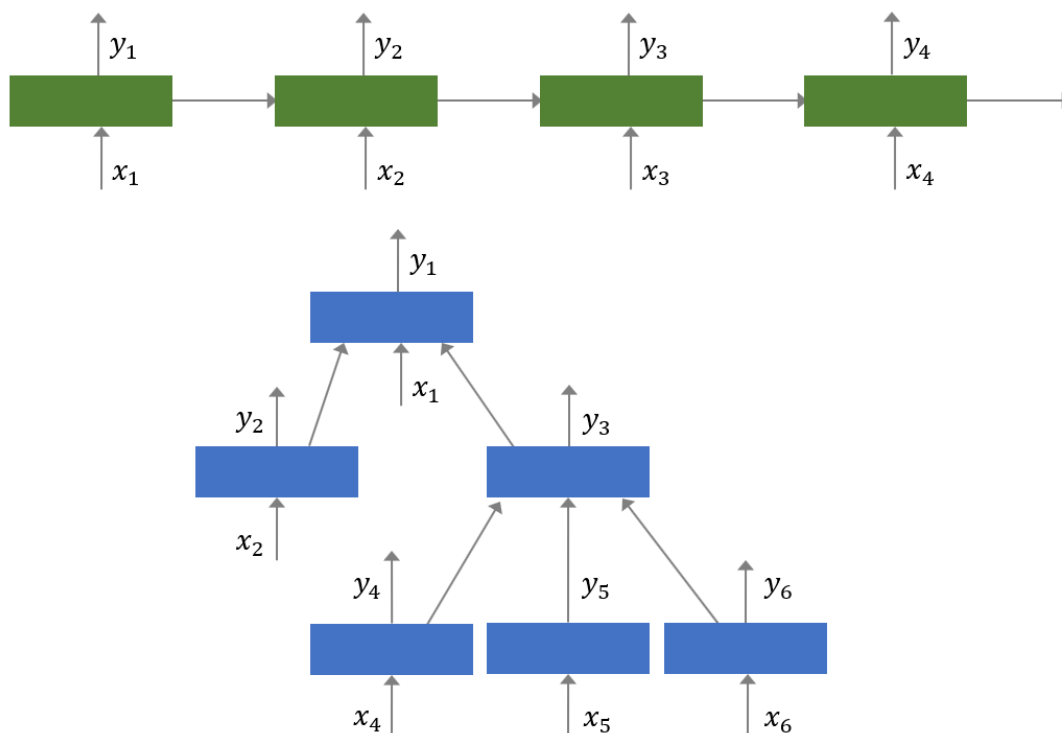


Figura 14: Comparação entre a arquitetura de uma LSTM, acima, e uma Tree-LSTM, abaixo. (Reproduzido de (TAI; SOCHER; MANNING, 2015))

Uma representação em árvore da consulta é gerada da seguinte forma: todas as propriedades (relações) existentes na consulta são convertidas em nós pai, e os filhos desses nós consistem em variáveis ou recursos (entidades). Para ilustrar isso, considere o exemplo mostrado na Figura 9, onde várias consultas foram geradas para responder à pergunta: “*Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?*” A árvore que representa a consulta que responde corretamente a essa pergunta teria a propriedade “*dbp:champions*” na raiz, com a entidade “*dbr:Ayrton_Senna*” e a variável “*?resp*” como seus nós filhos, como mostrado na Figura 15. Caso a consulta tivesse mais de uma tripla, o processo seria repetido, começando pelo nó que não representa uma variável, nesse caso, “*dbr:Ayrton_Senna*”, o que envolveria a substituição do nó “*dbr:Ayrton_Senna*” pelo elemento da nova tripla que representasse uma relação, fazendo com que o nó “*dbr:Ayrton_Senna*” se tornasse assim um nó folha da árvore.

As representações em árvore da pergunta e da consulta SPARQL são submetidas individualmente a uma Tree-LSTM para obter-se um *embedding* que representa a árvore.

?resp dbp:champions dbr:Ayrton_Senna

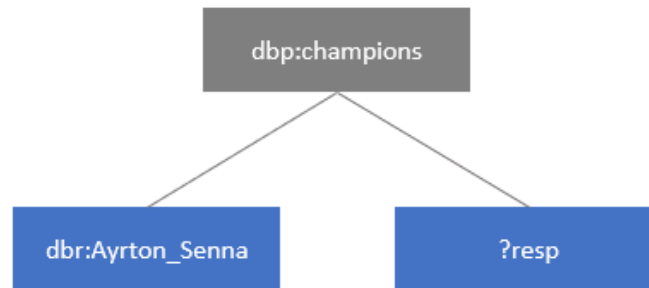


Figura 15: Representação em árvore de uma tripla de uma consulta SPARQL que responde à pergunta: “Em quais campeonatos Ayrton Senna foi campeão de Fórmula 1?”

De posse do *embedding* da árvore representando a pergunta e do *embedding* da árvore representando a consulta, ambos são submetidos a uma NN que irá calcular um grau de similaridade levando em conta a distância e o ângulo entre os dois vetores. Na Figura 16 podemos observar um diagrama ilustrativo do método de cálculo de similaridade. Ao final do processo, as consultas com maior similaridade com a pergunta de entrada são melhores ranqueadas e assim, escolhidas para serem executadas.

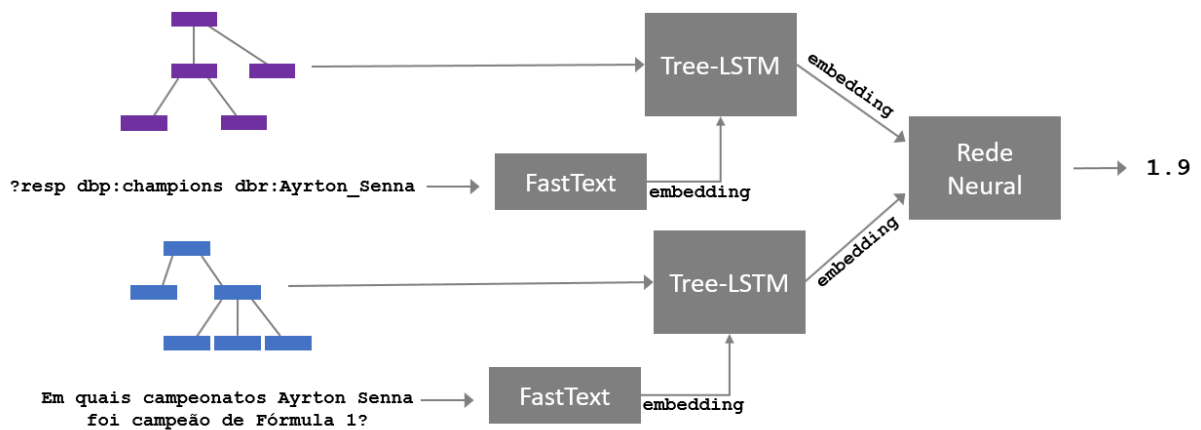


Figura 16: Processo de cálculo da similaridade entre uma pergunta e a sua respectiva consulta em SPARQL.

5 Avaliação Experimental

Neste capítulo abordaremos os experimentos realizados e os seus respectivos resultados, iniciando pela definição dos datasets utilizados. Após isso, serão mostrados os resultados dos classificadores utilizados para o Classificador do Tipo de Pergunta e para o Classificador de Relação Dupla, além dos resultados do treinamento da Tree-LSTM. Por fim serão mostrados os experimentos e resultados da solução completa.

5.1 Datasets

Avaliamos nossa abordagem com dois *benchmarks* padrão de KGQA: QALD e LCQuAD. O **QALD** ([USBECK et al., 2017](#)) é um *dataset* que faz parte de uma iniciativa da comunidade científica para promover o desenvolvimento de sistemas de KGQA. Ele inclui uma diversidade de perguntas e idiomas, uma variedade de dados vinculados e atualizações periódicas. A versão QALD-7 consiste em 215 perguntas e suas respectivas consultas SPARQL e possui traduções em oito idiomas diferentes, incluindo português, inglês, espanhol, italiano. O **LCQuAD** ([TRIVEDI et al., 2017](#)) é um *dataset* de KGQA que foi criado para endereçar perguntas complexas. Atualmente ele possui duas versões: LCQuAD v1, com 5.000 exemplos de perguntas em inglês e suas respectivas consultas SPARQL utilizando apenas o DBPedia como o GC, e LCQuAD v2, contendo 30.000 exemplos, abrangendo consultas tanto para a DBPedia quanto para a Wikidata.

Neste trabalho o dataset LCQuAD v1 foi utilizado para o treinamento do modelo de Tree-LSTM utilizado no módulo de Ranqueamento das Consultas, visto que era o *dataset* que possuía uma quantidade maior de exemplos (5000). Para o treinamento da Tree-LSTM o LCQuAD foi dividido em treino (70%), validação (20%) e teste (10%). Como esse *dataset* só possui originalmente perguntas em inglês, foi realizada a tradução das perguntas para o português utilizando a API do Google Translate. Além disso, tanto o LCQuAD quanto o QALD foram utilizados separadamente para o treinamento dos modelos de Classificação do Tipo de Pergunta, com o objetivo de escolher o modelo que

obtivesse o melhor desempenho entre os dois *datasets*. Para o treinamento do Classificador de Tipo de Pergunta, os *datasets* foram divididos em treino (80%) e teste (20%). Já para o teste da solução final proposta, KGQA_{PT}, foram utilizados três conjuntos de teste diferentes: (i) conjunto de teste do LCQuAD, contendo 500 exemplos, (ii) todo o LCQuAD, contendo 5000 exemplos e (iii) todo o QALD-7, contendo 215 exemplos.

5.2 Métricas de Avaliação

As métricas *Precision*, *Recall* e F1-Score são métricas comuns usadas para avaliar o desempenho de sistemas de recuperação de informação, classificação e outras tarefas de aprendizado de máquina. Elas são particularmente úteis em contextos em que a distribuição de classes é desigual, como em tarefas de QA, por exemplo. A métrica ***Precision*** mede a proporção de exemplos positivos previstos corretamente (*True Positives* (TP)) em relação ao conjunto total de instâncias previstas como positivas $TP + FP$. Ela é útil quando o custo dos falsos positivos (*False Positives* (FP)), é alto, ou seja, quando queremos garantir que o que foi classificado como positivo seja realmente positivo. Em termos simples, é a qualidade das previsões positivas e sua fórmula é dada por:

$$Precision = \frac{TP}{TP + FP} \quad (5.1)$$

O ***Recall*** mede a proporção de exemplos positivos previstos corretamente TP em relação ao total de instâncias que são realmente positivas $TP + FN$. Ela é útil quando o custo dos falsos negativos (*False Negatives* (FN)), é alto, ou seja, quando queremos garantir que todas as instâncias positivas sejam identificadas. A fórmula do *Recall* é dada por:

$$Recall = \frac{TP}{TP + FN} \quad (5.2)$$

O **F1-Score** é a média harmônica entre *Precision* e *Recall*. Ele fornece um equilíbrio entre essas duas métricas e é especialmente útil quando há um desbalanceamento entre as classes previstas. Ele atinge seu valor máximo quando tanto *Precision* quanto *Recall* são maximizados. Se um deles for baixo, o F1-Score será reduzido. O F1 é muito útil para avaliar a eficácia de algoritmos em tarefas como KGQA, onde é importante equilibrar a

capacidade de recuperar respostas corretas (*Recall*) com a precisão na identificação dessas respostas (*Precision*). Sua fórmula é dada por:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5.3)$$

5.3 Resultados do Classificador do Tipo de Pergunta

Um dos principais componentes do sistema KGQA_{PT} é o Classificador do Tipo de Pergunta, que envolve classificar a pergunta de entrada em um dos três tipos: Lista, Contagem ou Booleano. A sua grande importância vem do fato de que é ele quem define o formato da consulta a ser gerada. Por exemplo, se a pergunta for do tipo Booleano, a consulta SPARQL que responde corretamente à pergunta começará com a cláusula **ASK**. Já no caso de perguntas do tipo Lista, a consulta deverá iniciar pela cláusula **SELECT** e, para perguntas do tipo contagem, a consulta gerada precisa conter as cláusulas **SELECT** e **COUNT()**.

Foram testados três modelos para a classificação da pergunta de entrada: *Support Vector Machine* (SVM), *Random Forest* (RF) e Rede Neural (RN). Cada modelo foi treinado de duas formas diferentes: uma delas utilizando como *feature* de entrada o vetor de TF/IDF da pergunta lematizada e a outra utilizando o *embedding* médio das palavras lematizadas do texto de entrada. Além disso, os modelos foram treinados tanto no QALD-7 quanto no LCQuAD, com o objetivo de escolher o modelo com o melhor desempenho entre os dois *datasets*. Como podemos ver na Tabela 1, para o *dataset* LCQuAD com TF/IDF, os desempenhos dos três modelos foram muito parecidos, com destaque para o RF que atingiu o melhor desempenho nos três tipos de pergunta: lista, booleano e contagem, registrando um F1-macro de 99.4%.

Tabela 1: Classificador do Tipo de Pergunta com **TF/IDF** para o *dataset* **LCQuAD**

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Lista	99.5	99.7	99.4	99.8	99.9	99.7	99.6	99.8	99.6
Booleano	98.6	99.2	97.3	97.8	98.1	97.0	98.2	98.6	97.1
Contagem	99.8	100.0	99.2	97.9	99.2	97.7	98.8	99.6	98.5
<i>Média macro</i>	99.3	99.6	98.6	98.52	99.1	98.1	98.9	99.4	98.4

Na Tabela 2, observamos que, para o *dataset* QALD com TF/IDF, novamente o RF obteve o melhor desempenho na classificação dos três tipos de pergunta com um F1-macro de 79.3%, embora seja digno de nota que ocorreu uma queda de desempenho geral em relação ao modelo com TF/IDF no LCQuAD exibido na Tabela 1.

Tabela 2: Classificador do Tipo de Pergunta com **TF/IDF** para o *dataset* **QALD**

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Lista	93.3	97.0	92.5	93.3	91.6	89.4	93.3	94.3	90.9
Booleano	89.5	77.4	66.7	58.6	82.8	55.2	70.8	80.0	60.4
Contagem	41.2	46.7	38.9	100.0	100.0	100.0	58.3	63.6	56.0
<i>Média macro</i>	74.7	73.7	66.0	83.9	91.5	81.5	74.2	79.3	69.1

Utilizando o *embedding* como *feature* para os modelos, observamos que, para o *dataset* LCQuAD, o modelo RN obteve o melhor desempenho na classificação dos três tipos de pergunta, com um F1-macro de 98.2%, conforme pode ser visto na Tabela 3.

Tabela 3: Classificador do Tipo de Pergunta com *embedding* para o *dataset* LCQuAD

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Lista	98.3	95.7	99.4	99.7	100.0	99.6	99.0	97.8	99.6
Booleano	97.7	100.0	97.5	93.2	81.5	97.0	95.4	89.2	97.3
Contagem	99.0	100.0	98.3	93.2	83.0	97.4	96.0	90.7	97.9
<i>Média macro</i>	98.3	98.6	98.4	95.3	88.2	98.0	96.8	92.8	98.2

Já para o *dataset* QALD utilizando como *feature* o *embedding* do texto lematizado da pergunta, o melhor desempenho ficou com o modelo SVM, com F1-macro de 70.1%, mas vale ressaltar que em comparação com os modelos treinados com o LCQuAD, o desempenho geral ficou muito abaixo.

Tabela 4: Classificador do Tipo de Pergunta com *embedding* para o *dataset* QALD

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Lista	94.1	84.1	92.2	88.3	97.8	86.0	91.1	90.4	89.0
Booleano	73.1	0.0	75.0	65.5	0.0	51.7	69.1	0.0	61.2
Contagem	33.3	28.6	25.0	100.0	28.6	100.0	50.0	28.6	40.0
<i>Média macro</i>	66.8	37.6	64.1	84.6	42.1	79.3	70.1	39.7	63.4

Concluimos assim que para o Classificador do Tipo de Pergunta, os modelos treinados com o *dataset* LCQuAD obtiveram os melhores resultados no geral, o que pode ser explicado pelo tamanho maior do *dataset* LCQuAD (5000) em relação ao QALD (215). Em relação às *features* utilizadas, o TF/IDF obteve o melhor desempenho geral em relação ao *embedding*. Já dentre os modelos testados, o modelo RF obteve os maiores valores de F1.

Sendo assim, o modelo escolhido para a Classificação do Tipo de Pergunta foi o *Random Forest*, treinado no LCQuAD com TF/IDF que registrou um F1-macro de 99.4%.

5.4 Classificador de Relação Dupla

Como destacado na seção 4.4, em alguns casos, uma relação identificada no texto da pergunta inicial deve ser utilizada mais de uma vez no processo de geração da consulta para que a pergunta seja respondida corretamente. Para resolver esse problema, foi desenvolvido um Classificador de Relação Dupla para o qual foram testados três algoritmos: RF, SVM e RN e dois tipos de entrada: TF/IDF e vetor de *embedding* médio do texto. Cada pergunta é classificada como tendo ou não relação dupla, caso a pergunta seja classificada como tendo relação dupla, uma nova tripla é adicionada à lista de triplas já criada, para permitir o uso de uma mesma relação mais de uma vez. Somente o *dataset* LCQuAD foi utilizado para treinamento, uma vez que possui perguntas mais complexas e uma quantidade maior de relações duplas nas consultas. O dataset LCQuAD, contendo 5000 exemplos, foi dividido em dois conjuntos: treino (80%) e teste (20%).

Conforme podemos ver na Tabela 5, usando TF/IDF como *feature*, o modelo RN obteve o melhor desempenho de classificação, levando em conta as duas classes: Relação única e Relação dupla, registrando um F1-macro de 61.5%. Já para o experimento onde o *embedding* é utilizado como *feature*, o SVM e a RN atingiram resultados bem parecidos, mas o modelo RN com *embedding* obteve o melhor resultado F1-macro, alcançando 73.7% e, por isso, foi escolhido como o modelo para Classificação de Relação Dupla.

Tabela 5: Classificador de Relação Dupla com **TF/IDF** para o *dataset* **LCQuAD**

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Relação única	90.8	90.8	92.2	99.7	99.3	95.7	95.0	94.9	93.9
Relação dupla	62.5	45.5	37.1	5.2	5.2	24.0	9.6	9.3	29.1
<i>Média macro</i>	76.7	68.1	64.7	52.4	52.3	59.8	52.3	52.1	61.5

Tabela 6: Classificador de Relação Dupla com *embedding* para o *dataset* **LCQuAD**

	Precision			Recall			F1-Score		
	SVM	RF	RN	SVM	RF	RN	SVM	RF	RN
Relação única	93.9	90.5	94.7	97.7	100.0	95.6	95.8	95.0	95.2
Relação dupla	65.0	100.0	54.5	40.6	1.0	50.0	50.0	2.1	52.2
<i>Média macro</i>	79.5	95.2	74.6	69.2	50.5	72.8	72.9	48.5	73.7

5.5 Tree-LSTM

O modelo Tree-LSTM foi treinado utilizando o *dataset* LCQuAD-1, que contém 5000 exemplos. O *dataset* foi dividido em três conjuntos: treino (70%), validação (20%) e teste (10%). A anotação dos graus de similaridade no *dataset* ocorreu da seguinte forma: para consultas que levam à resposta correta, o grau de similaridade entre a pergunta e a consulta é igual a 2, caso contrário, é igual a 1. Utilizando o conjunto de validação, os hiper-parâmetros foram otimizados, de forma que os parâmetros finais escolhidos para o modelo de Tree-LSTM são listados na Tabela 7.

Tabela 7: Hiper-parâmetros para o modelo Tree-LSTM

Parâmetro	Valor
Dimensões da entrada	300×1
Dimensão da memória da LSTM	150×1
Quantidade de épocas	15
Tamanho de Mini Batch	25
Taxa de aprendizado	1×10^{-2}
Regularização (<i>Weight Decay</i>)	2.25×10^{-3}
<i>Dropout</i>	0.2
Função de custo	Divergência de Kullback-Leibler
Otimizador	Adagrad

5.6 KGQA_{PT}

A solução final envolve a execução de cada um dos 5 módulos sequencialmente, ou seja, primeiramente extraímos as *features* sintáticas da pergunta de entrada através do módulo Analisador Sintático. Após isso, classificamos o tipo de pergunta utilizando o componente Classificador de Tipo de Pergunta e extraímos as entidades e relações pelo módulo Mapeamento de Conceitos. De posse do tipo de pergunta e das entidades e relações, criamos as consultas através do módulo Geração de Consultas e, por fim, ranqueamos as consultas geradas através do Ranqueamento de Consultas.

Realizamos o teste da solução final em três *datasets*: conjunto de teste do LCQuAD, contendo 500 exemplos, conjunto completo do LCQuAD, contendo 5000 exemplos e conjunto completo do QALD, que possui 215 exemplos. Para cada *dataset*, fizemos um teste com e sem o classificador de relações PTRL implementado neste trabalho, com o objetivo de verificar o desempenho da solução final com o seu uso. Por conta da falta de modelos para a tarefa de KGQA em português que poderiam servir de *baseline*, criamos um *baseline* traduzindo as perguntas do QALD do português para o inglês e submetendo ao modelo QASPARQL, em inglês, proposto em (LIANG et al., 2021). Esse teste só foi possível para o *dataset* QALD, já que é o único que possui perguntas originalmente em português. Na Tabela 8 é possível observar os resultados do QASPARQL com perguntas traduzidas.

Tabela 8: Avaliação do QASPARQL no *dataset* QALD-7

	Precision	Recall	F1-Score
Lista	87.7	18.4	30.5
Booleano	100.0	3.5	6.7
Contagem	100.0	42.9	60.0
Tudo	91.1	16.9	28.5

Em um primeiro experimento, realizamos o teste da solução final no *dataset* QALD-7 sem o uso do PTRL, obtendo um F1-score de 35.3 %. Conforme podemos ver na Tabela 9, o tipo de pergunta booleano obteve o pior resultado, com um F1 de apenas 18.8%.

Tabela 9: Avaliação do **KGQA_{PT}** no *dataset* **QALD-7** sem PTRL

	Precision	Recall	F1-Score
Lista	80.5	24.7	37.8
Booleano	100	10.34	18.8
Contagem	100	28.6	44.4
Tudo	82.6	22.4	35.3

Em seguida, foi realizado um experimento utilizando o conjunto de teste do LCQuAD, contendo 500 exemplos, sem o uso do PTRL. Vale ressaltar que os dados presentes nesse *dataset* não foram utilizados para o treinamento de nenhum modelo de nenhum dos componentes da solução. Nesse experimento o F1-score foi de 29.5%, onde novamente o tipo booleano obteve o pior desempenho de 13.3%, conforme podemos ver na Tabela 10.

Tabela 10: Avaliação do **KGQA_{PT}** no conjunto de teste do *dataset* **LCQuAD** sem PTRL

	Precision	Recall	F1-Score
Lista	62.5	26.8	37.5
Booleano	75.0	7.3	13.3
Contagem	50.0	18.0	26.5
Tudo	59.3	19.7	29.5

Completando os testes sem o uso do PTRL, realizamos um experimento utilizando todo o *dataset* LCQuAD, contendo 5000 exemplos. Embora uma parte do *dataset* tenha sido utilizado como conjunto de treino para o Classificador do Tipo de Pergunta e também para o treinamento da Tree-LSTM, o que poderia indicar um vazamento do conjunto de treino para o conjunto de teste, esse experimento serve para verificarmos como a solução final lida com um *dataset* de perguntas complexas contendo uma quantidade maior de exemplos. O resultado do F1 do modelo para o LCQuAD sem PTRL foi de 43.4%, com

destaque para o tipo booleano que novamente não performou bem, registrando F1 de 15.5%, como pode ser visto na Tabela 11.

Tabela 11: Avaliação do **KGQA_{PT}** em todo o *dataset* **LCQuAD** sem PTRL

	Precision	Recall	F1-Score
Lista	65.1	22	32.9
Booleano	93.9	8.5	15.5
Contagem	54.6	60.3	57.3
Tudo	63.6	32.9	43.4

Conforme destacado anteriormente, por conta da escassez de modelos de Ligação de Relação para o português, desenvolvemos neste trabalho o modelo PTRL. Com objetivo de verificar qual a influência do uso do PTRL na performance do sistema KGQA_{PT}, realizamos alguns experimentos com o seu uso. Na Tabela 12, podemos verificar que usando o PTRL no dataset QALD-7, obtivemos um F1 de 41.9%, um resultado 6.6% acima do experimento utilizando o QALD-7 sem o uso do PTRL.

Tabela 12: Avaliação do **KGQA_{PT}** no *dataset* **QALD-7** com PTRL

	Precision	Recall	F1-Score
Lista	80.5	32.4	46.2
Booleano	75	10.3	18.2
Contagem	66.7	28.6	40
Tudo	79.4	28.5	41.9

Em relação ao conjunto de teste do LCQuAD, contendo 500 exemplos, e com o uso do PTRL, o resultado foi de um F1 de 35.4%, um aumento de 5.9% em relação ao experimento com o LCQuAD de teste sem o uso do PTRL, conforme podemos visualizar na Tabela 13.

Tabela 13: Avaliação do **KGQA_{PT}** no conjunto de teste do *dataset* **LCQuAD** com PTRL

	Precision	Recall	F1-Score
Lista	73	32.6	45
Booleano	85.7	14.3	24.5
Contagem	43.5	19.7	27
Tudo	64.2	24.5	35.4

Por fim, em relação ao conjunto completo do LCQuAD, contendo 5000 exemplos, a performance do KGQA_{PT} foi de 45.8%, um acréscimo de 2.4% em relação ao modelo sem o uso do PTRL, conforme visto na Tabela 14.

Tabela 14: Avaliação do **KGQA_{PT}** em todo o *dataset* **LCQuAD** com PTRL

	Precision	Recall	F1-Score
Lista	65.9	25	36.3
Booleano	93.2	11.2	19.9
Contagem	55.4	62.2	58.6
Tudo	64.5	35.5	45.8

Na Figura 17, podemos visualizar um comparativo do desempenho do KGQA_{PT} com e sem o uso do PTRL. Conforme podemos observar, houve um ganho de performance da solução final com uso do PTRL em todos os *datasets*, mostrando que a utilização do PTRL realmente melhorou o desempenho do sistema como um todo.

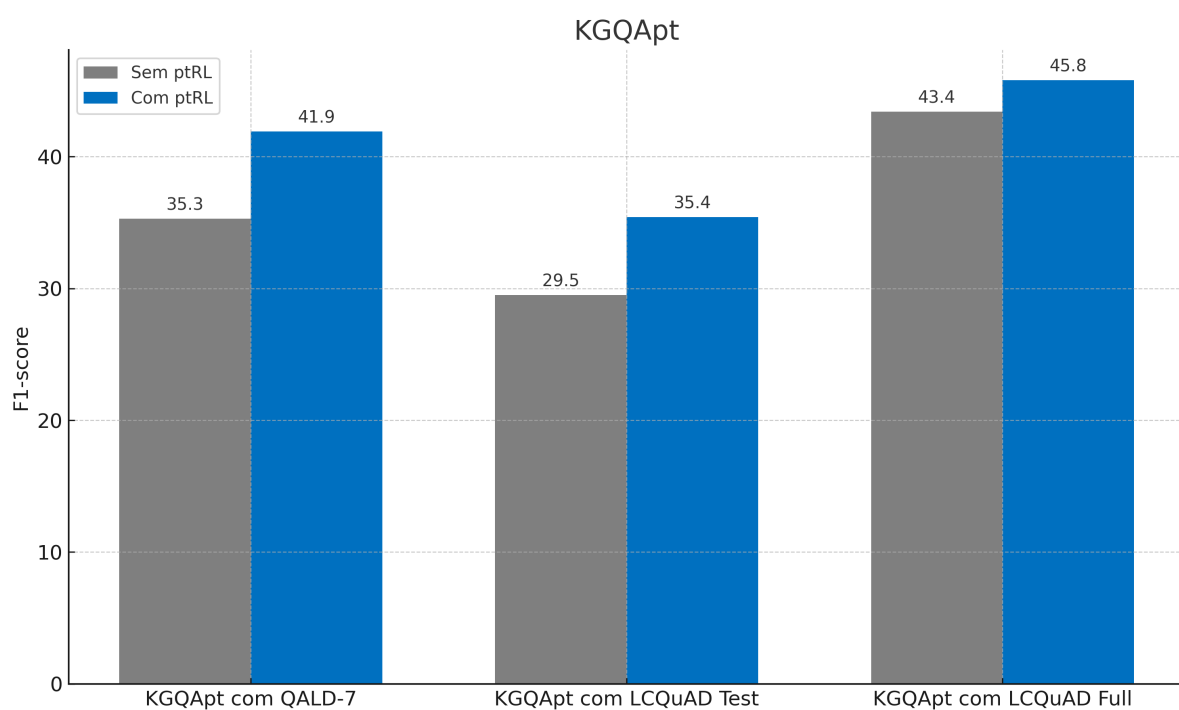


Figura 17: Comparativo do desempenho do **KGQA_{PT}** com e sem o uso do modelo de Ligação de Relações **ptRL**

6 Conclusão

Neste trabalho, propomos uma solução da tarefa de KGQA para o português, chamada de *Portuguese Knowledge Graph Question Answering* (KGQA_{PT}), um sistema modular, composto de 5 componentes. O primeiro módulo é responsável pela extração de *features* sintáticas que são utilizadas pelos módulos seguintes. O segundo módulo realiza a classificação do tipo de pergunta, enquanto o terceiro é responsável por realizar o mapeamento de entidades e relações da pergunta para o Grafo de Conhecimento. O quarto componente realiza a construção das consultas SPARQL, possíveis candidatas à resposta da pergunta. O quinto e último módulo realiza o ranqueamento das consultas geradas a fim de selecionar aquela com maior probabilidade de responder corretamente à pergunta.

Mostramos que foi possível resolver a tarefa de KGQA para o português, com um resultado de F1-score de 45.8% no LCQuAD e 41.9% no QALD, valor muito superior ao *baseline* adotado, o que responde à primeira questão de pesquisa: “*Qual é o desempenho de um sistema baseado em componentes que aborda a tarefa de KGQA para a língua portuguesa?*”. Por conta da escassez de recursos que realizem a tarefa de Ligação de Relação, crucial para o sucesso do sistema, criamos o PTRL, um sistema de Ligação de Relação para o português. O desempenho do PTRL foi avaliado ao comparar a performance do KGQA_{PT} com e sem o uso do PTRL, o que mostrou que o uso do PTRL ocasionou um ganho de 6.6% de F1 no QALD e de 2.4% no LCQuAD, comprovando a eficácia do PTRL, o que responde à outra questão de pesquisa: “*Qual é o impacto de alguns componentes no desempenho do modelo?*” Este trabalho contribui para a pesquisa na área de PLN em português, uma vez que cria e disponibiliza novas ferramentas e métodos para a tarefa de KGQA em português e estabelece um *baseline* para essa tarefa, possibilitando que novos modelos que venham a ser desenvolvidos tenham um parâmetro de comparação.

Algumas limitações que devem ser consideradas na interpretação dos resultados. Em primeiro lugar, este trabalho não incorporou modelos customizados para LE e somente um para LR, potencialmente influenciando o desempenho do sistema nessas tarefas específicas. Além disso, o sistema concentra-se exclusivamente em três tipos de consultas

(Lista, Booleano e Contagem), que podem não abranger toda a gama de tipos de consulta encontrados em cenários do mundo real. Estas limitações destacam áreas para pesquisas futuras e a necessidade de uma avaliação abrangente utilizando diversos conjuntos de dados.

Como trabalhos futuros, sugerimos o desenvolvimento de novos métodos de Ligação de Entidade e de Ligação de Relação para o português, de forma a aumentar a quantidade de modelos utilizados pelo módulo de Mapeamento de Conceitos e diminuir a quantidade de perguntas sem entidades ou relações mapeadas. Por fim, vale destacar que este trabalho gerou um artigo intitulado: “*Can SPARQL Talk in Portuguese? Answering Questions in Natural Language Using Knowledge Graphs*”, que foi aceito na *16th International Conference on Computational Processing of Portuguese*. (PROPOR 2024).

REFERÊNCIAS

ABU-SALIH, Bilal. Domain-specific knowledge graphs: A survey. **Journal of Network and Computer Applications**, Elsevier, v. 185, p. 103076, 2021. DOI: [10.1016/j.jnca.2021.103076](https://doi.org/10.1016/j.jnca.2021.103076).

AMARAL, Carlos et al. Priberam's question answering system for Portuguese. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 4022 LNCS, p. 410–419, 2006. DOI: [10.1007/11878773_46](https://doi.org/10.1007/11878773_46).

AUER, Sören et al. DBpedia: A nucleus for a Web of open data. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 4825 LNCS, p. 722–735, 2007. DOI: [10.1007/978-3-540-76298-0_52](https://doi.org/10.1007/978-3-540-76298-0_52).

AYOOLA, Tom et al. ReFinED: An Efficient Zero-shot-capable Approach to End-to-End Entity Linking. **NAACL 2022 - 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Industry Papers**, p. 209–220, 2022. DOI: [10.48550/arXiv.2207.04108](https://doi.org/10.48550/arXiv.2207.04108).

BARBARESI, Adrien. **Simplemma**. [S. l.]: Zenodo, jan. 2023. DOI: [10.5281/zenodo.7555188](https://doi.org/10.5281/zenodo.7555188).

BERTRAM, Niels; DUNKEL, Jürgen; HERMOSO, Ramón. I am all EARS: Using open data and knowledge graph embeddings for music recommendations. **Expert Systems with Applications**, v. 229, 2023. DOI: [10.1016/j.eswa.2023.120347](https://doi.org/10.1016/j.eswa.2023.120347).

BICK, Eckhard. **The parsing system palavras: Automatic grammatical analysis of Portuguese in a constraint grammar framework**. [S. l.]: Aarhus Universitetsforlag, 2000.

BIRD, Steven; KLEIN, Ewan; LOPER, Edward. **Natural language processing with Python: analyzing text with the natural language toolkit**. [S. l.]: O'Reilly Media, Inc., 2009.

- BOJANOWSKI, Piotr et al. Enriching word vectors with subword information. **Transactions of the association for computational linguistics**, MIT Press, v. 5, p. 135–146, 2017.
- BOLLACKER, Kurt et al. Freebase: a collaboratively created graph database for structuring human knowledge. In: PROCEEDINGS of the 2008 ACM SIGMOD international conference on Management of data. [S. l.: s. n.], 2008. p. 1247–1250. DOI: [10.1145/1376616.1376746](https://doi.org/10.1145/1376616.1376746).
- BRANK, Janez; LEBAN, Gregor; GROBELNIK, Marko. Annotating documents with relevant wikipedia concepts. **Proceedings of SiKDD**, v. 472, 2017.
- CARVALHO, Gracinda; DE MATOS, David Martins; ROCIO, Vitor. IdSay: Question answering for portuguese. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 5706 LNCS, p. 345–352, 2009. DOI: [10.1007/978-3-642-04447-2_40](https://doi.org/10.1007/978-3-642-04447-2_40).
- CARVALHO, Nuno Ramos; SIMÕES, Alberto; ALMEIDA, José João. Bootstrapping a data-set and model for question-answering in Portuguese. **OpenAccess Series in Informatics**, v. 94, 2021. DOI: [10.4230/OASICS.SLATE.2021.18](https://doi.org/10.4230/OASICS.SLATE.2021.18).
- CHEN, Yanji; KOKAR, Mieczyslaw M.; MOSKAL, Jakub J. SPARQL Query Generator (SQG). **Journal on Data Semantics**, v. 10, n. 3-4, p. 291–307, 2021. DOI: [10.1007/s13740-021-00133-y](https://doi.org/10.1007/s13740-021-00133-y).
- COSTA, Luis. First evaluation of esfinge - A question answering system for portuguese. **Lecture Notes in Computer Science**, v. 3491, p. 522–533, 2005. DOI: [10.1007/11519645_51](https://doi.org/10.1007/11519645_51).
- DAIBER, Joachim et al. Improving efficiency and accuracy in multilingual entity extraction. **ACM International Conference Proceeding Series**, p. 121–124, 2013. DOI: [10.1145/2506182.2506198](https://doi.org/10.1145/2506182.2506198).
- DUBEY, Mohnish et al. EARL: joint entity and relation linking for question answering over knowledge graphs. In: SPRINGER. INTERNATIONAL Semantic Web Conference. [S. l.: s. n.], 2018. p. 108–126. DOI: [10.1007/978-3-030-00671-6_7](https://doi.org/10.1007/978-3-030-00671-6_7).
- EBERHARD, David M; SIMONS, Gary F; FENNIG, Charles D. *Ethnologue: Languages of the world*(22nd edn.). Dallas, TX: SIL International. <http://www.ethnologue.com>. Acessado em: 01/09/2023.

- FÄRBER, Michael. The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 11779 LNCS, p. 113–129, 2019. DOI: [10.1007/978-3-030-30796-7_8](https://doi.org/10.1007/978-3-030-30796-7_8).
- GOYAL, Palash; PANDEY, Sumit; JAIN, Karan. **Deep learning for natural language processing**. [S. l.]: Springer, 2018.
- HAGIWARA, Masato. **Real-World Natural Language Processing: Practical Applications with Deep Learning**. [S. l.]: Simon e Schuster, 2021.
- HOGAN, Aidan et al. Knowledge Graphs. **CoRR**, abs/2003.02320, 2020. arXiv: [2003.02320](https://arxiv.org/abs/2003.02320).
- HONNIBAL, Matthew et al. spaCy: Industrial-strength Natural Language Processing in Python, 2020. DOI: [10.5281/zenodo.1212303](https://doi.org/10.5281/zenodo.1212303).
- INTERNET WORLD STATS. **Internet World Stats**. [S. l.: s. n.]. <https://www.internetworldstats.com/stats7.htm>. Acessado em: 01/09/2023.
- JOULIN, Armand et al. Bag of Tricks for Efficient Text Classification. In: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers**. Valencia, Spain: Association for Computational Linguistics, abr. 2017. p. 427–431.
- JUN, Changwook et al. ANNA: Enhanced Language Representation for Question Answering. **Proceedings of the Annual Meeting of the Association for Computational Linguistics**, p. 121–132, 2022.
- KACUPAJ, Endri et al. VQuAnDa: Verbalization QUestion ANswering DATaset. In: **THE Semantic Web**. [S. l.]: Springer International Publishing, 2020. p. 531–547.
- KETSMUR, Maksym; RODRIGUES, Mário; TEIXEIRA, António. A question and answer system for factual queries in Portuguese on DBPEDIA. **Proceedings of the International Conference on WWW/Internet 2017 and Applied Computing 2017**, p. 87–94, 2017.
- LIANG, Shiqi et al. Querying knowledge graphs in natural language. **Journal of Big Data**, v. 8, n. 1, 2021. DOI: [10.1186/s40537-020-00383-w](https://doi.org/10.1186/s40537-020-00383-w).
- MENDES, Pablo N. et al. DBpedia spotlight: Shedding light on the web of documents. **ACM International Conference Proceeding Series**, p. 1–8, 2011. DOI: [10.1145/2063518.2063519](https://doi.org/10.1145/2063518.2063519).

- MOMTAZI, Saeedeh; ABBASIANAEI, Zahra. **Question Answering over Text and Knowledge Base**. [S. l.: s. n.], 2022. p. 1–202. DOI: [10.1007/978-3-031-16552-8](https://doi.org/10.1007/978-3-031-16552-8).
- NAVIGLI, Roberto; PONZETTO, Simone Paolo. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. **Artificial Intelligence**, v. 193, p. 217–250, 2012. DOI: [10.1016/j.artint.2012.07.001](https://doi.org/10.1016/j.artint.2012.07.001).
- OLAH, Christopher. **Understanding LSTM Networks**. [S. l.: s. n.]. <https://colah.github.io/posts/2015-08-Understanding-LSTMs>. Acessado em 01/10/2023.
- OLIVEIRA, Hugo Gonçalo; COELHO, Inês; GOMES, Paulo. Exploiting Portuguese lexical knowledge bases for answering open domain cloze questions automatically. **Proceedings of the 9th International Conference on Language Resources and Evaluation, LREC 2014**, p. 4202–4209, 2014.
- PEREVALOV, A. et al. Knowledge Graph Question Answering Leaderboard: A Community Resource to Prevent a Replication Crisis. **2022 Language Resources and Evaluation Conference, LREC 2022**, p. 2998–3007, 2022.
- PURKAYASTHA, Sukannya et al. A Deep Neural Approach to KGQA via SPARQL Silhouette Generation. **Proceedings of the International Joint Conference on Neural Networks**, 2022-July, 2022. DOI: [10.1109/IJCNN55064.2022.9892263](https://doi.org/10.1109/IJCNN55064.2022.9892263).
- QUARESMA, Paulo; RODRIGUES, Irene. A question-answering system for Portuguese juridical documents. **Proceedings of the International Conference on Artificial Intelligence and Law**, p. 256–257, 2005. DOI: [10.1145/1165485.1165536](https://doi.org/10.1145/1165485.1165536).
- RAJPURKAR, Pranav et al. **SQuAD: 100,000+ Questions for Machine Comprehension of Text**. [S. l.: s. n.], 2016. arXiv: [1606.05250 \[cs.CL\]](https://arxiv.org/abs/1606.05250).
- RAVI, Manoj Prabhakar Kannan et al. CHOLAN: A modular approach for neural entity linking on wikipedia and wikidata. **EACL 2021 - 16th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference**, p. 504–514, 2021.
- RONY, Md Rashad Al Hasan et al. SGPT: A Generative Approach for SPARQL Query Generation from Natural Language Questions. **IEEE Access**, v. 10, p. 70712–70723, 2022. DOI: [10.1109/ACCESS.2022.3188714](https://doi.org/10.1109/ACCESS.2022.3188714).
- SAIAS, José; QUARESMA, Paulo. The Senso question answering approach to Portuguese QA@CLEF-2007. **CEUR Workshop Proceedings**, v. 1173, 2007.

- SAKOR, Ahmad; SINGH, Kuldeep; VIDAL, Maria-Esther. FALCON: an entity and relation linking framework over dbpedia. In: AACHEN: RWTH. CEUR Workshop Proceedings 2456 (2019). [S. l.: s. n.], 2019. v. 2456, p. 265–268.
- SANTOS, Diana; ROCHA, Paulo. The key to the first CLEF with Portuguese: Topics, questions and answers in CHAVE. **Lecture Notes in Computer Science**, v. 3491, p. 821–832, 2005. Cited by: 20; All Open Access, Green Open Access. DOI: [10.1007/11519645_80](https://doi.org/10.1007/11519645_80).
- SHAVARANI, Hassan S.; SARKAR, Anoop. **SpEL: Structured Prediction for Entity Linking**. [S. l.: s. n.], 2023. arXiv: [2310.14684](https://arxiv.org/abs/2310.14684) [cs.CL].
- SINGH, K; RADHAKRISHNA, A S; BOTH, A; SHEKARPOUR, S; LYTRA, I; USBECK, R; VYAS, A; KHIKMATULLAEV, A; PUNJANI, D; LANGE, C et al. Why reinvent the wheel: Let’s build question answering systems together. In: THE Web Conference 2018 - Proceedings of the World Wide Web Conference, WWW 2018. [S. l.: s. n.], 2018. p. 1247–1256. DOI: [10.1145/3178876.3186023](https://doi.org/10.1145/3178876.3186023).
- SINGH, Kuldeep; RADHAKRISHNA, Arun Sethupat; BOTH, Andreas; SHEKARPOUR, Saeedeh; LYTRA, Ioanna; USBECK, Ricardo; VYAS, Akhilesh; KHIKMATULLAEV, Akmal; PUNJANI, Dharmen; LANGE, Christoph et al. Why reinvent the wheel: Let’s build question answering systems together. In: PROCEEDINGS of the 2018 world wide web conference. [S. l.: s. n.], 2018. p. 1247–1256.
- SINGHAL, Amit. **Introducing the Knowledge Graph: Things, not Strings**. [S. l.: s. n.], mai. 2012. <https://blog.google/products/search/introducing-knowledge-graph-things-not/>. Acessado em: 18/01/2023.
- SOUSA, Alysson Gomes de et al. Using a domain ontology to bridge the gap between user intention and expression in natural language queries. **ICEIS 2020 - Proceedings of the 22nd International Conference on Enterprise Information Systems**, v. 1, p. 751–758, 2020.
- STENGEL, Markus. **Strings of Natural Languages: Unsupervised Analysis and Segmentation on the Expression Level**. [S. l.]: Diplom. de, 2007.
- SUCHANEK, Fabian M; KASNECI, Gjergji; WEIKUM, Gerhard. Yago: a core of semantic knowledge. In: PROCEEDINGS of the 16th international conference on World Wide Web. [S. l.: s. n.], 2007. p. 697–706.

TAI, Kai Sheng; SOCHER, Richard; MANNING, Christopher D. Improved semantic representations from tree-structured long short-Term memory networks. **ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference**, v. 1, p. 1556–1566, 2015. DOI: [10.3115/v1/p15-1150](https://doi.org/10.3115/v1/p15-1150).

TAN, Yiming et al. **Can ChatGPT Replace Traditional KBQA Models? An In-depth Analysis of the Question Answering Performance of the GPT LLM Family**. [S. l.: s. n.], 2023. arXiv: [2303.07992](https://arxiv.org/abs/2303.07992) [cs.CL].

THAMBI, S.V.; REGHURAJ, P.C. Towards Improving the Performance of Question Answering System using Knowledge Graph - A Survey. **Proceedings of the 2nd International Conference on Artificial Intelligence and Smart Energy, ICAIS 2022**, p. 672–679, 2022. cited By 0. DOI: [10.1109/ICAIS53314.2022.9742802](https://doi.org/10.1109/ICAIS53314.2022.9742802).

THANAKI, Jalaj. **Python natural language processing**. [S. l.]: Packt Publishing Ltd, 2017.

TOPICS, Exploding. **How Much Data is Generated Every Day?** [S. l.: s. n.]. <https://explodingtopics.com/blog/data-generated-per-day>. Acessado em 01/09/2023.

TRIVEDI, Priyansh et al. LC-QuAD: A corpus for complex question answering over knowledge graphs. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 10588 LNCS, p. 210–218, 2017. DOI: [10.1007/978-3-319-68204-4_22](https://doi.org/10.1007/978-3-319-68204-4_22).

USBECK, Ricardo et al. 7th open challenge on question answering over linked data (QALD-7). **Communications in Computer and Information Science**, v. 769, p. 59–69, 2017. DOI: [10.1007/978-3-319-69146-6_6](https://doi.org/10.1007/978-3-319-69146-6_6).

VRANDEČIĆ, Denny; KRÖTZSCH, Markus. Wikidata: a free collaborative knowledgebase. **Communications of the ACM**, ACM New York, NY, USA, v. 57, n. 10, p. 78–85, 2014.

W3C SEMANTIC WEB STANDARDS. **SPARQL 1.1 Overview**. [S. l.: s. n.], 2023. <https://www.w3.org/TR/sparql11-overview/>. Acessado em: 01/09/2023.

YAMADA, Ikuya et al. LUKE: Deep contextualized entity representations with entity-aware self-attention. **EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference**, p. 6442–6454, 2020.

- YANG, Zhilin et al. XLNet: Generalized autoregressive pretraining for language understanding. **Advances in Neural Information Processing Systems**, v. 32, 2019.
- ZAFAR, Hamid; NAPOLITANO, Giulio; LEHMANN, Jens. Formal query generation for question answering over knowledge bases. In: SPRINGER. EUROPEAN semantic web conference. [S. l.: s. n.], 2018. p. 714–728.
- ZAREMBA, Wojciech; SUTSKEVER, Ilya. **Learning to Execute**. [S. l.: s. n.], 2015. arXiv: [1410.4615](https://arxiv.org/abs/1410.4615) [cs.NE].
- ZHANG, Long; WU, Tianxing et al. Auto Insurance Knowledge Graph Construction and Its Application to Fraud Detection. **ACM International Conference Proceeding Series**, p. 64–70, 2021. DOI: [10.1145/3502223.3502231](https://doi.org/10.1145/3502223.3502231).
- ZHANG, Wenzheng; HUA, Wenyue; STRATOS, Karl. ENTQA: ENTITY LINKING AS QUESTION ANSWERING. **ICLR 2022 - 10th International Conference on Learning Representations**, 2022.