



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CENTRO DE CIÊNCIAS**  
**DEPARTAMENTO DE COMPUTAÇÃO**  
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**  
**DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO**

**CAIO VIKTOR DA SILVA AVILA**

**UM FRAMEWORK INDEPENDENTE DE DOMÍNIO PARA KNOWLEDGE GRAPH**  
**QUESTION ANSWERING BASEADO EM LARGE LANGUAGE MODELS**

**FORTALEZA**

**2024**

CAIO VIKTOR DA SILVA AVILA

UM FRAMEWORK INDEPENDENTE DE DOMÍNIO PARA KNOWLEDGE GRAPH  
QUESTION ANSWERING BASEADO EM LARGE LANGUAGE MODELS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência da Computação.

Orientadora: Prof.<sup>a</sup> Dr.<sup>a</sup> Vânia Maria Ponte Vidal.

Coorientador: Prof. Dr. Marco Antonio Casanova.

FORTALEZA

2024

Dados Internacionais de Catalogação na Publicação  
Universidade Federal do Ceará  
Sistema de Bibliotecas  
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

A972f Avila, Caio Viktor da Silva.

Um framework independente de domínio para knowledge graph question answering baseado em large language models / Caio Viktor da Silva Avila. – 2024.  
134 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2024.

Orientação: Profa. Dra. Vânia Maria Ponte Vidal.

Coorientação: Profa. Dra. Marco Antonio Casanova.

1. question answering. 2. knowledge graph. 3. large language model. 4. rdf browser. I. Título.

CDD 005

---

CAIO VIKTOR DA SILVA AVILA

UM FRAMEWORK INDEPENDENTE DE DOMÍNIO PARA KNOWLEDGE GRAPH  
QUESTION ANSWERING BASEADO EM LARGE LANGUAGE MODELS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência da Computação.

Aprovada em: 12 de Setembro de 2024.

BANCA EXAMINADORA

---

Prof.<sup>a</sup> Dr.<sup>a</sup> Vânia Maria Ponte  
Vidal (Orientadora)  
Universidade Federal do Ceará (UFC)

---

Prof. Dr. Marco Antonio  
Casanova (Coorientador)  
Pontifícia Universidade Católica do Rio de  
Janeiro (PUC-Rio)

---

Prof.<sup>a</sup> Dr.<sup>a</sup> Melissa Lemos Cavalière  
Pontifícia Universidade Católica do Rio de  
Janeiro (PUC-Rio)

---

Prof. Dr. Altigran Soares da Silva  
Universidade Federal do Amazonas (UFAM)

---

Prof. Dr. José Maria Da Silva Monteiro Filho  
Universidade Federal do Ceará (UFC)

---

Prof. Dr. José Wellington Franco da Silva  
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

## AGRADECIMENTOS

À Instituição Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro com a manutenção da bolsa de auxílio. À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Capes, pelo custeio de inscrições em congressos acadêmicos.

À minha família por todo o suporte e carinho fornecidos durante as horas de dificuldades.

Aos Professores Dr<sup>a</sup>. Vânia Maria Ponte, Dr. Marco Antônio Casanova e Dr. José Wellington Franco da Silva pela excelente orientação.

Aos professores participantes da banca examinadora Dr<sup>a</sup>. Melissa Lemos Cavalière e Dr. José Maria Monteiro e Dr. Altigran Soares da Silva pelo tempo e valiosas sugestões.

Aos colegas da turma de doutorado, pelas reflexões, críticas e sugestões recebidas. Especialmente ao meu colega de doutorado Túlio Vidal Rolim, pelas contribuições em coautorias de artigos.

"Qualquer tecnologia suficientemente avançada  
é equivalente à mágica." (Arthur C. Clarke,  
1973, p. xx.)

## RESUMO

Os *knowledge graph question answering* (KGQA) são sistemas computacionais capazes de responder perguntas em linguagem natural utilizando um *knowledge graph* (KG) como fonte de conhecimento a ser consultada. Estes sistemas destacam-se por suas respostas curadas e profundas. Ao longo da história, diversas arquiteturas e abordagens foram propostas para sistemas de KGQA, com sistemas baseados em modelos fim-a-fim pré-treinados de aprendizado profundo vindo a se popularizar nos últimos anos. Atualmente, os *large language models* (LLMs) apresentam-se como o estado da arte para modelos de linguagem pré-treinados. Deste modo, surge a oportunidade do desenvolvimento de sistemas de KGQA baseados em LLMs. Com isto em vista, como principal contribuição, esta tese apresenta Auto-KGQA, um *framework* autônomo independente de domínio baseado em LLMs para KGQA. O *framework* seleciona automaticamente sub-grafos do KG que são relevantes para a questão, que o LLM utiliza como contexto para traduzir a pergunta em linguagem natural para uma consulta SPARQL sobre o KG. O *framework* é acessível através de sua API HTTP ou por meio de uma interface Web de *Chat Messenger*. Além disso, o *framework* é integrado ao RDF *browser*, LiRB, permitindo a navegação iterativa de recursos retornados em consultas. Experimentos preliminares com Auto-KGQA com o ChatGPT que indicam o *framework* reduziu substancialmente o número de *tokens* passados para o LLM sem sacrificar o desempenho. Por fim, a avaliação de Auto-KGQA em um *benchmark* com consultas empresariais no domínio de companhias de seguros mostrou que o *framework* é competitivo, alcançando uma melhoria de 13,2% na acurácia em relação ao estado da arte e uma de redução de 51,12% na quantidade de *tokens* repassados ao LLM. Experimentos revelaram que o uso de estratégias de *few-shot learning* em conjunto com o sub-grafo selecionado por Auto-KGQA geram sistemas de KGQA robustos e generalizáveis, superando seus competidores em cenários *0-shot learning* e equiparando-se em cenários *few-shot*.

**Palavras-chave:** question answering; knowledge graph; large language model; rdf browser.

## ABSTRACT

Knowledge graph question answering (KGQA) systems are computational systems capable of answering questions in natural language using a knowledge graph (KG) as a source of knowledge to be consulted. These systems stand out for their curated and deep answers. Throughout history, several architectures and approaches have been proposed for KGQA systems, with systems based on pre-trained end-to-end deep learning models becoming popular in recent years. Currently, large language models (LLMs) are the state of the art for pre-trained language models. Thus, the opportunity arises to develop KGQA systems based on LLMs. With this in mind, as its main contribution, this thesis presents Auto-KGQA, a domain-independent autonomous framework based on LLMs for KGQA. The framework automatically selects fragments of the KG that are relevant to the question, which the LLM uses as context to translate the natural language question into a SPARQL query over the KG. The framework is accessible through its HTTP API or through a Chat Messenger Web interface. In addition, the framework is integrated with the RDF browser, LiRB, allowing iterative navigation of resources returned in queries. Preliminary experiments with Auto-KGQA with ChatGPT indicate that the framework substantially reduced the number of tokens passed to LLM without sacrificing performance. Finally, evaluation of Auto-KGQA on a benchmark with enterprise queries in the insurance companies domain showed that the framework is competitive, achieving a 13.2% improvement in accuracy over the state-of-the-art and a 51.12% reduction in the number of tokens passed to LLM. Experiments have revealed that the use of *few-shot learning* strategies together with the subgraph selected by Auto-KGQA generates robust and generalizable KGQA systems, outperforming their competitors in *0-shot learning* scenarios and matching them in *few-shot* scenarios.

**Keywords:** question answering; knowledge graph; large language model; rdf browser.

## LISTA DE FIGURAS

Figura 1 – Etapas de consulta realizadas pelo Auto-KGQA . . . . .	15
Figura 2 – Tarefas no processamento de <i>Question Answering</i> (QA) . . . . .	20
Figura 3 – Fluxo de sistemas de QA end-to-end. . . . .	21
Figura 4 – <i>KG on Prompt</i> . . . . .	25
Figura 5 – <i>IR on Prompt</i> . . . . .	27
Figura 6 – <i>Text on Prompt</i> . . . . .	28
Figura 7 – <i>KG Embeddings</i> . . . . .	29
Figura 8 – Quantidade de trabalhos utilizando cada dataset . . . . .	42
Figura 9 – Arquitetura do Framework Auto-KGQA . . . . .	45
Figura 10 – Workflow do Framework Auto-KGQA. . . . .	46
Figura 11 – Passos da etapa offline realizados pelo Auto-KGQA . . . . .	49
Figura 12 – Passos da etapa online realizados pelo Auto-KGQA . . . . .	61
Figura 13 – Interface Web de Chat com o Auto-KGQA. . . . .	81
Figura 14 – Tela de seleção de classes. . . . .	84
Figura 15 – Tela de navegação de recurso. . . . .	85
Figura 16 – Representação de como a relação “Works At” é armazenada. . . . .	86
Figura 17 – Visualização de Timeline. . . . .	88
Figura 18 – Resultado de consulta salva. . . . .	89
Figura 19 – Visualização do recorte do grafo de consulta salva. . . . .	89
Figura 20 – Esquema do KG utilizado nos experimentos. . . . .	95
Figura 21 – Redução percentual no número de <i>tokens</i> passados por questão comparado ao KG com e sem inferências. . . . .	103
Figura 22 – Número de conceitos perdidos no sub-grafo para cada uma das perguntas erradas. . . . .	106
Figura 23 – Recall para perguntas respondidas corretamente. . . . .	107

## LISTA DE TABELAS

Tabela 1 – Consolidação dos dados extraídos dos trabalhos estudados. . . . .	41
Tabela 2 – Tipos de eventos de timeline . . . . .	87
Tabela 3 – Questões utilizadas para avaliar o ChatGPT . . . . .	92
Tabela 4 – Questões utilizadas para avaliar o ChatGPT . . . . .	93
Tabela 5 – Questões utilizadas para avaliar o ChatGPT . . . . .	94
Tabela 6 – Resumo de resultados de experimentos. . . . .	96
Tabela 7 – Quantidade de tokens passados pelo Auto-KGQA por questão. . . . .	102
Tabela 8 – Comparação entre resultados de Auto-KGQA e Sequeda et al. . . . .	105
Tabela 9 – Comparação de tokens do KG usados na entrada . . . . .	105
Tabela 10 – Análise dos erros nas consultas do benchmark . . . . .	109
Tabela 11 – Análise dos erros nas consultas do benchmark parte 2 . . . . .	110
Tabela 12 – Resultados do experimentos no dataset SciQA . . . . .	114
Tabela 13 – F1-Score de Auto-KGQA por tipo de questão . . . . .	115
Tabela 14 – Publicações relacionadas ao tema principal de pesquisa . . . . .	127
Tabela 15 – Publicações secundárias . . . . .	127

## LISTA DE ABREVIATURAS E SIGLAS

EKG	<i>Enterprise Knowledge Graph</i>
IR	<i>Information Retrieval</i>
KB	<i>Knowledge Base</i>
KBQA	<i>Knowledge Base Question Answering</i>
KG	<i>Knowledge Graph</i>
KGQA	<i>Knowledge Graph Question Answering</i>
KIR	<i>Keyword-Based Information Retrieval</i>
LLM	<i>Large Language Model</i>
LN	<i>Linguagem Natural</i>
ML	<i>Machine Learning</i>
NER	<i>Named Entity Recognition</i>
OWL	<i>Web Ontology Language</i>
QA	<i>Question Answering</i>
QPG	<i>Questões de Pesquisa Gerais</i>
RDB	<i>Relational Databases</i>
RDF	<i>Resource Description Framework</i>
RDFS	<i>Resource Description Framework Schema</i>
SPARQL	<i>SPARQL Protocol and RDF Query Language</i>
SQA	<i>Semantics Question Answering</i>
SQL	<i>Structured Query Language</i>
SW	<i>Semantic Web</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>14</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>18</b>
<b>2.1</b>	<b>Ontologias</b>	<b>18</b>
<b>2.2</b>	<i>Knowledge Graphs</i>	<b>19</b>
<b>2.3</b>	<b>Sistemas de Question Answering (QA)</b>	<b>19</b>
<b>2.4</b>	<b>Sistemas de QA fim-a-fim</b>	<b>21</b>
<b>2.5</b>	<b>Vetor de embeddings</b>	<b>22</b>
<b>2.6</b>	<b>Graph Embeddings</b>	<b>23</b>
<b>2.7</b>	<b>Large Language Models</b>	<b>24</b>
<b>2.7.1</b>	<i>Abordagens de Passagem do KG para o LLM</i>	<b>25</b>
<b>2.7.1.1</b>	<i>Graph on Prompt</i>	<b>25</b>
<b>2.7.1.2</b>	<i>IR on Prompt</i>	<b>26</b>
<b>2.7.1.3</b>	<i>Text on Prompt</i>	<b>28</b>
<b>2.7.1.4</b>	<i>KG embeddings</i>	<b>29</b>
<b>2.8</b>	<b>In-context Learning</b>	<b>30</b>
<b>2.9</b>	<i>Model fine-tuning</i>	<b>31</b>
<b>2.10</b>	<b>ChatGPT</b>	<b>32</b>
<b>2.11</b>	<b>RDF Browsers</b>	<b>33</b>
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>34</b>
<b>3.1</b>	<b>Abordagens para seleção do sub-grafo relevante para perguntas</b>	<b>34</b>
<b>3.2</b>	<b>Abordagens de KGQA baseadas em LLMs</b>	<b>36</b>
<b>3.3</b>	<b>Dataset e métricas utilizadas para avaliar sistemas de KGQA baseados em LLMs</b>	<b>42</b>
<b>4</b>	<b>AUTO-KGQA: UM FRAMEWORK BASEADO EM LLM PARA KGQA</b>	<b>45</b>
<b>4.1</b>	<b>Framework Outline</b>	<b>46</b>
<b>4.2</b>	<b>Etapa Offline</b>	<b>48</b>
<b>4.2.1</b>	<i>Extração do T-Box mínimo</i>	<b>49</b>
<b>4.2.2</b>	<i>Geração de Índices do KG</i>	<b>53</b>
<b>4.2.2.1</b>	<i>GenerateIndexes Algorithm</i>	<b>54</b>
<b>4.2.2.2</b>	<i>PopulateIndex Algorithm</i>	<b>55</b>

4.2.3	<i>Geração de Índice de Exemplos</i>	60
4.3	<b>Etapa Online</b>	61
4.3.1	<i>Algoritmo para Responder Questões em Linguagem Natural</i>	62
4.3.2	<i>Algoritmo de Extração do Sub-grafo Relevante para a Questão</i>	62
4.3.2.1	<i>Algoritmo para Identificar Citações a Recursos do KG na Questão</i>	67
4.3.2.2	<i>Algoritmo para Recuperar Potenciais Triplas Relevantes no endpoint</i>	68
4.3.2.3	<i>Algoritmo de Filtragem de Triplas Relevantes</i>	69
4.3.3	<i>Algoritmo de Text-to-SPARQL</i>	75
4.3.4	<i>Algoritmo de Geração da Resposta Final em Linguagem Natural</i>	77
4.3.5	<i>Templates de Prompts</i>	78
4.4	<b>Interface Web de Chat</b>	80
5	<b>LIGHT RDF BROWSER</b>	82
5.1	<b>LiRB: Light RDF Browser</b>	83
5.1.1	<i>Navegação de Recursos</i>	84
5.1.2	<i>Timelines de Recursos</i>	87
5.1.3	<i>Consultas Salvas</i>	88
6	<b>RESULTADOS E DISCUSSÕES</b>	91
6.1	<b>Daily-life Dataset</b>	91
6.1.1	<i>Experimentos Manuais</i>	96
6.1.2	<i>Experimentos com Auto-KGQA</i>	100
6.1.3	<i>Experimentos com Langchain</i>	101
6.1.4	<i>Redução no número de Tokens passados</i>	102
6.2	<b>Resultados do Auto-KGQA em um <i>benchmark</i> de questões empresariais em linguagem natural</b>	103
6.3	<b>SciQA Dataset</b>	113
6.4	<b>Discussões sobre os Resultados</b>	123
7	<b>CONCLUSÃO</b>	128
	<b>REFERÊNCIAS</b>	130

## 1 INTRODUÇÃO

Sistemas de QA são sistemas computacionais que são capazes de responder questões realizadas em Linguagem Natural (LN) (HIRSCHMAN; GAIZAUSKAS, 2001). Um sistema de QA tem como objetivo interpretar uma pergunta feita por um humano e responde-la de maneira natural e sucinta, dispensando a necessidade do uso de linguagens de consulta formais, tais como *Structured Query Language (SQL)* e *SPARQL Protocol and RDF Query Language (SPARQL)*. Este tipo de sistema possui uma curva de aprendizado suave, o que expande o acesso à informação para usuários não-técnicos.

Dentre os inúmeros tipos de sistemas QA existentes, um tipo destaca-se por suas respostas curadas e profundas, os *Knowledge Base Question Answering (KBQA)* ou *Knowledge Graph Question Answering (KGQA)*. Sistemas de KGQA são sistemas de QA que possuem um *Knowledge Graph (KG)* como fonte de conhecimento a ser consultada. Estes sistemas traduzem uma consulta em linguagem natural para uma consulta em linguagem formal de consulta suportada pelo KG em questão, tal como SPARQL (YANI; KRISNADHI, 2021).

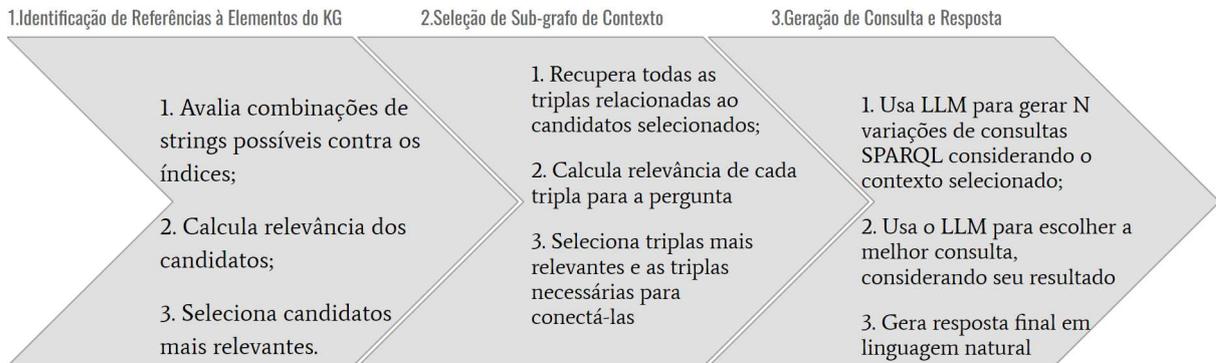
Ao longo da história, diversas arquiteturas e abordagens foram propostas para sistemas de QA, com sistemas baseados em modelos fim-a-fim pré-treinados de aprendizado profundo vindo a se popularizar nos últimos anos (DIEFENBACH *et al.*, 2018). O principal impulsionador no desenvolvimento de sistemas de QA fim-a-fim foi o desenvolvimento de redes baseadas na arquitetura de *transformers*, tais como o BERT e seus afins (VASWANI *et al.*, 2017).

Atualmente, os *Large language models (LLMs)* apresentam-se como o estado da arte para modelos de linguagem pré-treinados. LLMs são modelos generativos genéricos com um grande número de parâmetros treinados para a tarefa de completar textos (CHANG *et al.*, 2023). LLMs são treinados em um grande conjunto de dados estruturados e não-estruturados de uma grande variedade de domínios. Este tipo de modelo destaca-se por sua grande variedade de aplicações, alcançando ótimos resultados em um grande número de tarefas, dentre as quais encontram-se *Question and Answering* e escrita de consultas SQL e SPARQL (BROWN *et al.*, 2020).

Deste modo, surge a oportunidade do desenvolvimento de sistemas de KGQA baseados em LLMs (TAN *et al.*, 2023). Tendo isto em vista, esta tese busca investigar qual a abordagem mais apropriada para o uso de LLMs na tarefa de KGQA, resultando no desenvolvimento de um *framework* para KGQA baseado em LLM. Experimentos sugerem que a melhor abordagem consiste em traduzir a questão em LN para uma consulta SPARQL, considerando

tanto o T-Box, quanto o A-Box. No entanto, passar ao LLM o T-Box e o A-Box completos pode ser inviável em cenários do mundo real.

Figura 1 – Etapas de consulta realizadas pelo Auto-KGQA



Fonte: elaborada pelo próprio autor.

Portanto, nesta tese é apresentado Auto-KGQA, um *framework* autônomo independente de domínio baseada em LLMs para KGQA. Conceitualmente, o algoritmo seguido por Auto-KGQA é composto por três etapas apresentadas na Figura 1: (1) Identificação de Referências à Elementos do KG; (2) Seleção de Sub-grafo de Contexto ; (3) Geração de Consulta e Resposta. Dado um KG  $K$  e uma pergunta em LN  $Q$ , o *framework* seleciona fragmentos do T-Box e A-Box de  $K$ , gerando um sub-grafo  $K'$  composto por triplas relevantes para  $Q$  que será utilizado como contexto para o LLM traduzir  $Q$  para uma consulta SPARQL sobre  $K$ . Auto-KGQA gera  $n$  consultas SPARQL,  $S_1, S_2, \dots, S_n$ , que traduzem  $Q$  e seleciona a melhor, denominando-a  $S$ , com base nos resultados da execução de cada consulta sobre  $K$ . Finalmente, Auto-KGQA gera uma resposta em LN para o usuário, com base na questão  $Q$ , consulta  $S$ , e o resultado da execução de  $S$  sobre  $K$ . Ao melhor de meu conhecimento, Auto-KGQA é o primeiro sistema de KGQA capaz de selecionar de maneira autônoma  $K'$ , em contraste aos demais sistemas que utilizam técnicas de *fine-tuning* ou já recebem  $K'$  como entrada. Deste modo, a principal inovação de Auto-KGQA é o algoritmo de seleção de fragmentos menores do KG suficientes para realizar a escrita da consulta, reduzindo assim o número de *tokens* passados como entrada para o LLM, o que diminui a possibilidade de alucinações com elementos irrelevantes na entrada e aumenta a capacidade de suportar grandes KGs.

Como um recurso adicional, o *framework* tem suporte ao uso de exemplos de consultas já conhecidas para auxiliar o LLM na tarefa de geração de consultas SPARQL. Nesta abordagem o sistema recebe como entrada pares de perguntas em linguagem natural e suas respectivas consultas SPARQL, buscando em tempo de execução quais destas são similares à

consulta sendo respondida. Além disso, o sistema registra o histórico de questões, consultas SPARQL e respostas geradas, colhendo do usuário se a resposta foi correta ou não. Para os casos julgados corretos, o sistema utiliza os pares de perguntas e suas respectivas SPARQLs como futuros exemplos conhecidos.

Além disso, nesta tese é apresentado o uso do *RDF browser* LiRB para complementar a experiência do usuário durante a consulta ao KG. *Light RDF Browser* (LiRB) é uma interface leve para navegação interativa em KGs RDF. Nesta integração, recursos retornados através da interface Web de Chat podem ser utilizados como ponto de partida para uma navegação iterativa sobre o KG.

As Questões de Pesquisa Gerais (*Questões de Pesquisa Gerais* (QPG)s) a serem respondidas neste trabalho são:

- *QPG*<sub>1</sub>: Como responder questões em linguagem natural sobre um KG utilizando LLMs?
- *QPG*<sub>2</sub>: Como filtrar triplas relevantes para uma questão de um KG?
- *QPG*<sub>3</sub>: Quais os principais desafios para LLMs no cenário de KGQA?

De maneira resumida, as principais contribuições deste trabalho são listadas a baixo:

- Um *framework* baseado em LLMs para KGQA;
- Um algoritmo para extrair um sub-grafo relevante para a pergunta sendo respondida, considerando triplas de ambos, T-Box e A-Box.

Por sua vez, as contribuições secundárias são:

- Experimentação empírica do uso do ChatGPT como ferramenta de KGQA;
- Um *benchmark* composto por um KG no domínio de vida cotidiano e 15 pares de questões e respostas para avaliar o desempenho de sistemas de KGQA;
- Avaliação de desempenho do *framework* em diferentes *datasets* de domínio-fechado, tais como: (1) perguntas sobre o cotidiano; (2) perguntas empresariais sobre dados de companhias de seguro; (3) perguntas sobre publicações científicas, acompanhado de exemplos de consultas;
- Uma interface leve para a navegação interativa baseada em texto sobre KGs RDF.

O restante desta tese é estruturado como se segue. No capítulo 2 é apresentada uma fundamentação teórica necessária para compreender o restante desta tese. A seguir, o Capítulo 3 apresenta os trabalhos relacionados ao tema desta tese. Posteriormente, o capítulo 4 apresenta o *framework* Auto-KGQA para a realização autônoma independente de domínio de KGQA utilizando LLMs. O Capítulo 5 apresenta a ferramenta de navegação de KGs *Light RDF Browser*

(LiRB) integrada ao *framework*. Dando sequência, o Capítulo 6 apresenta os resultados e discussões dos experimentos realizados. Por fim, o Capítulo 7 apresenta as conclusões adquiridas nesta tese.

## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado um breve apanhado sobre temas envolvidos no desenvolvimento desta tese.

### 2.1 Ontologias

Segundo (GRUBER, 1993), uma Ontologia é uma especificação explícita de uma conceitualização formal. Em outras palavras, uma Ontologia é uma representação estruturada de conceitos existentes em um domínio de conhecimento específico de maneira inequívoca e interpretável por máquinas. No campo da *Semantic Web (SW)*, uma Ontologia pode ser implementada em linguagens como *Resource Description Framework Schema (RDFS)*<sup>1</sup> ou *Web Ontology Language (OWL)*<sup>2</sup>, seguindo o padrão *Resource Description Framework (RDF)*.

No campo da *SW*, Ontologias OWL são utilizadas para representar um conhecimento complexo acerca de um determinado domínio. Uma Ontologia OWL consiste de uma representação taxonômica de classes e propriedades. Esta representação modela o domínio como uma hierarquia de classes e propriedades, onde conceitos do mundo real são representados por classes, enquanto estas classes relacionam-se entre si através de propriedades.

OWL é uma linguagem baseada em lógica computacional, deste modo, o conhecimento expresso em OWL pode ser explorado por programas de computadores, *e.g.*, para verificar a consistência do conhecimento ou para tornar explícito o conhecimento implícito nos dados (W3, 2013).

Uma das características mais marcantes no uso de Ontologias para a representação dos dados é a capacidade da inferência de informações implícitas. Linguagens como OWL e *RDFS* possuem um conjunto de *Inference Rules* (Regras de Inferência) que permitem o descobrimento de novas informações de maneira automática, através da execução de algoritmos chamados Raciocinadores (*Reasoners*) (BERNERS-LEE *et al.*, 2001). Estes *Reasoners* utilizam um conjunto de regras específicas para cada linguagem para descobrir conhecimento implícito baseado nas informações já conhecidas na Ontologia, *e.g.*, caso a ontologia defina a existências de duas classes, **Classe A** e **Classe B**, onde **Classe B** é uma subclasse de **Classe A**, o *Reasoner* inferirá que todo recurso do tipo **Classe B** também é um recurso do tipo **Classe A**. Ontologias *RDFS* possuem um conjunto de regras de inferência muito reduzido, mas computacionalmente

---

<sup>1</sup> <https://www.w3.org/TR/rdf-schema/>

<sup>2</sup> <https://www.w3.org/OWL/>

barato, enquanto ontologias OWL possuem conjuntos muito mais complexos, mas mais caro computacionalmente.

Por conta do alto custo computacional do processo de inferência, OWL é dividido em subconjuntos de regimes de inferências ou perfis da linguagem, onde cada perfil possui um diferente poder de inferência e custo computacional<sup>3</sup>.

## 2.2 Knowledge Graphs

O termo Knowledge Graph (KG) ou Grafo de Conhecimento foi popularizado pelo *Google* em uma publicação em seu *blog* oficial intitulada “*Introducing the Knowledge Graph: things, not strings*”<sup>4</sup>. A empresa adotou o uso da tecnologia para aprimorar os resultados do seu motor de busca, adicionando-o pequenas “caixas” com informações estruturadas sobre o termo sendo buscando, além de sugerir entidades relacionadas a busca ligadas no KG. Inspiradas pelo *Google*, outras empresas passaram a investir na pesquisa e no uso dos KGs, e.g., *Graph Search* do *Facebook*<sup>5</sup>, o *Satori* da *Microsoft* (MICROSOFT, 2019), etc.

Um KG é um conjunto de dados estruturados que é compatível com o modelo de dados *RDF* e tem uma ontologia OWL como seu esquema (PAN *et al.*, 2017). Um KG não é necessariamente ligado a grafos de conhecimento externos; contudo, este tipo de informação pode ser útil para prover informações contextuais. Um ponto importante sobre KGs é que estes são esperados serem confiáveis, de alta qualidade, alta acessibilidade e serem capazes de prover um serviço de informação orientado ao usuário final (PAN *et al.*, 2017).

## 2.3 Sistemas de Question Answering (QA)

De acordo com (HIRSCHMAN; GAIZAUSKAS, 2001), os Sistemas de *Question Answering* (QA) são sistemas computacionais que são capazes de responder questões realizadas em LN. Um sistema de QA tem como objetivo interpretar uma pergunta feita por um humano e responde-la de maneira natural e sucinta, dispensando a necessidade do uso de linguagens de consulta formais, tais como *SQL* e *SPARQL*. Este tipo de sistema possui uma curva de aprendizado suave, o que expande o acesso à informação para usuários não-técnicos. Um sistema de QA pode atuar sobre uma grande variedade de fontes de dados, tais como um *corpus*,

<sup>3</sup> <https://www.w3.org/TR/sparql11-entailment/#OWL2-RDFBS-Profiles>

<sup>4</sup> <http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html>

<sup>5</sup> <https://www.facebook.com/graphsearcher/>

*Relational Databases (RDB)*s, KGs e outros tipos de *Knowledge Base (KB)*s atuando como uma camada de acesso aos dados de maneira rápida e prática. Segundo (DIEFENBACH *et al.*, 2018), sistemas de QA sobre bases de conhecimento (QAKB, *Question Answering over Knowledge Base*) tem como objetivo achar no KB/KG a informação solicitada via LN pelo usuário. Isto é realizado através da tradução da questão  $Q$  em LN para uma consulta  $Q'$  em SPARQL que a represente. Sistemas de QAKB também são conhecidos como sistemas de *Semantics Question Answering (SQA)*, KGQA ou KBQA (HÖFFNER *et al.*, 2017).

As principais diferenças entre um sistema de QA e um sistema de *Keyword-Based Information Retrieval (KIR)* são que enquanto um *KIR* é baseado na recuperação de documentos que contenham as *keywords* de entrada, um sistema de QA busca interpretar a questão da entrada e retornar sua resposta sucintamente (ALLAM; HAGGAG, 2012). Além disso, dependendo da base utilizada, tal como um *Enterprise Knowledge Graph (EKG)* como um KB, um QA pode responder questões as quais suas respostas não estão explicitamente presentes na base.

Figura 2 – Tarefas no processamento de QA



Fonte: (DIEFENBACH *et al.*, 2018).

Segundo o *survey* apresentado por (DIEFENBACH *et al.*, 2018), os principais sistemas de QA construídos até então seguem os seguintes passos (Figura 2):

1. **Análise de questão (*Question Analysis*):** Nesta etapa, a questão do usuário é analisada com base em características puramente sintáticas, realizando a segmentação da sentença, reconhecimento de entidades nomeadas (*Named Entity Recognition (NER)*) e a criação da árvore de dependência de frases;
2. **Mapeamento de frases (*Phrase Mapping*):** Nesta etapa, as frases (uma ou mais palavras da entrada) são mapeadas para recursos da base de conhecimento, tais como instâncias, classes ou propriedades;
3. **Desambiguação (*Desambiguation*):** Nesta etapa, são solucionadas possíveis ambiguidades das etapas anteriores, onde no passo 1 há a possibilidade de haver ambiguidade na segmentação ou na dependência entre segmentos, já no passo 2 uma mesma frase pode ser mapeada para múltiplos recursos;
4. **Construção de consulta (*Query Construction*):** Nesta etapa, é realizada a construção de

uma consulta em uma linguagem formal, e.g., SPARQL, que represente a interpretação da questão. Esta etapa pode ser realizadas das seguintes formas:

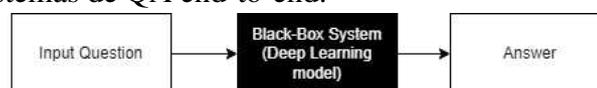
- **Baseada nas informações da análise de questão:** É criada uma árvore de análise sintática (*parse tree*) que é utilizada para a dedução da consulta SPARQL;
- **Baseada em *parser* semântico:** O sistema une as regras sintáticas da *parser tree* às regras semânticas, tais como hierarquia de classes, domínio e contra-domínio de propriedades;
- **Baseada em padrões (*Template-Based*):** O sistema tenta classificar a questão do usuário para padrões em linguagens formais de consultas, possuindo *slots* a serem preenchidos por parâmetros informados pelo usuário.

Contudo, embora sistemas de QA sejam capazes de abstrair a complexidade da linguagem de consulta, estes ainda necessitam que o usuário possua conhecimentos sobre o vocabulário da ontologia sendo consultada. Um dos principais desafios para o uso de sistemas de KGQA por usuários leigos no domínio é o chamado desafio da lacuna léxica (*Lexical Gap*). Lacuna léxica, também conhecido como lacuna de vocabulário (*Vocabulary Gap*), refere-se aos casos onde o vocabulário utilizado pelo usuário e o vocabulário usado para formalmente representar os dados podem ser substancialmente diferentes (HAKIMOV *et al.*, 2015). Isto pode ocorrer por conta de erros ortográficos e gramaticais, pelo uso de sinônimos ou quando o nível de abstração utilizado pelo usuário difere daquele modelado na fonte.

## 2.4 Sistemas de QA fim-a-fim

Arquiteturas de QAs end-to-end (QA fim-a-fim) tornaram-se o estado da arte, onde uma arquitetura mono-componente identifica a resposta com base em um conjunto de características geradas a partir da pergunta em linguagem natural, sem depender de componentes intermediários (SILVA *et al.*, 2020). Sistemas deste tipo são costumeiramente baseados em modelos de *deep-learning* (aprendizado profundo) da família *seq2seq*, onde o modelo atua como um sistema *black-box* (caixa-preta), tornando transparente o processo de geração da resposta.

Figura 3 – Fluxo de sistemas de QA end-to-end.



Fonte: Próprio autor.

Um modelo *Sequence-to-sequence* (*seq2seq*) é um modelo (rede neural) que tem

como entrada e saída uma sequência. Onde uma sequência é formada por um conjunto de símbolos (e.g., palavras) cuja ordem é relevante e tem um significado. Contudo, modelos de *Machine Learning* (ML) são preparados para trabalharem com entradas numéricas, geralmente representadas como vetores de tamanho fixo, o que não é o caso para sentenças em linguagem natural. Daí surge a necessidade de seguir o *framework encoder-decoder* (SUTSKEVER *et al.*, 2014). O *framework encoder-decoder* consiste no uso de duas camadas de modelos:

1. *encoder* (codificadora) que codifica a sequência de entrada (texto) em uma representação de vetor numérico de tamanho fixo;
2. *decoder* (decodificadora) que decodifica a saída numérica da rede em uma saída na representação final (texto).

## 2.5 Vetor de embeddings

*Embeddings* vetoriais (*embedding vectors*) ou vetores de *embeddings* são uma forma de converter palavras, frases e outros dados em números que capturam seus significados e relações. Eles representam diferentes tipos de dados como pontos em um espaço multidimensional, onde pontos de dados semelhantes são agrupados mais próximos. Essas representações numéricas ajudam as máquinas a compreender e processar esses dados de maneira mais eficaz.

*Embeddings* de palavras e frases são dois dos subtipos mais comuns de *embeddings* vetoriais, mas existem outros. Alguns *embeddings* vetoriais podem representar documentos inteiros, bem como vetores de imagem projetados para combinar conteúdo visual, vetores de perfil de usuário para determinar as preferências de um usuário, vetores de produto que ajudam a identificar produtos similares e muitos outros. Os *embeddings* vetoriais ajudam os algoritmos de *machine learning* a encontrar padrões nos dados e executar tarefas como análise de sentimentos, tradução de idiomas, sistemas de recomendação e muito mais.

*Embeddings* de palavras representam palavras individuais como vetores. Técnicas como Word2Vec (GOLDBERG; LEVY, 2014), GloVe (PENNINGTON *et al.*, 2014) e FastText (ATHIWARATKUN *et al.*, 2018) aprendem *embeddings* de palavras capturando relações semânticas e informações contextuais de grandes *corpora* de texto. As abordagens de *embeddings* de palavras (*word embeddings*), como Word2Vec e GloVe, já consideram as palavras no entorno durante o mapeamento de uma palavra para o espaço de *embedding*, levando em conta o contexto da palavra. Nestas abordagens, uma mesma palavra pode ser mapeada para vetores diferentes dependendo do contexto (palavras no entorno) em que está inserida.

*Embeddings* de frases representam frases inteiras como vetores. Modelos como *Universal Sentence Encoder* (USE) (CER *et al.*, 2018) e *SkipThought* (KIROS *et al.*, 2015) geram *embeddings* que capturam o significado geral e o contexto das frases.

## 2.6 Graph Embeddings

Um *embedding* de um grafo ou *graph embeddings* é um mapeamento de seus elementos (vértices e arestas) para um espaço vetorial, de modo que este preserve alguma medida de proximidade definida no grafo original (GOYAL; FERRARA, 2018). Em abordagens de *embeddings* de palavras, uma mesma palavra pode ser mapeada para vetores diferentes dependendo do contexto (palavras no entorno) em que está inserida. A aplicação de *embeddings* em grafos segue uma lógica semelhante.

As técnicas de *embedding* de grafos visam capturar as características estruturais e semânticas dos grafos, preservando relações importantes como conectividade e proximidade entre os vértices. Métodos avançados, como Node2Vec (GROVER; LESKOVEC, 2016) e DeepWalk (PEROZZI *et al.*, 2014), estendem a ideia de *word embeddings* para grafos, utilizando estratégias de caminhadas aleatórias (*random walks*) para gerar sequências de vértices análogas a sequências de palavras. Essas sequências são então processadas por algoritmos de *embedding*, que produzem representações vetoriais que mantêm as propriedades do grafo original.

Um desenvolvimento significativo na área de *embeddings* de grafos é o surgimento das *Graph Neural Networks* (GNNs). As GNNs são uma classe de redes neurais projetadas especificamente para operar em dados de grafos, permitindo a aprendizagem de representações de nós e arestas de forma fim-a-fim. Técnicas como *Graph Convolutional Networks* (GCNs) (KIPF; WELING, 2016), *Graph Attention Networks* (GATs) (VELICKOVIC *et al.*, 2017), e *GraphSAGE* (HAMILTON *et al.*, 2017) utilizam convoluções e mecanismos de atenção para agregar informações dos nós vizinhos, capturando assim tanto as características dos nós quanto a estrutura do grafo. A capacidade das GNNs de integrar características dos nós com a estrutura global do grafo permite que elas capturem dependências complexas e interações entre os elementos do grafo, oferecendo uma abordagem robusta e flexível para a modelagem de dados relacionais.

Os *embeddings* de grafos têm mostrado grande potencial em diversas aplicações, como a detecção de comunidades, a previsão de links, e a classificação de nós. Além disso, essas representações vetoriais permitem a aplicação de técnicas de aprendizado de máquina tradicionais

em dados de grafos, facilitando a análise e a interpretação de grandes estruturas complexas. A capacidade de capturar a essência das relações entre vértices e arestas em um espaço vetorial contínuo abre novas possibilidades para a exploração de dados estruturados e a extração de conhecimento em áreas como redes sociais, bioinformática e sistemas de recomendação.

## 2.7 Large Language Models

*Large Language Model* (LLM) (Modelos de Linguagem de Grande Escala) representam uma evolução significativa no campo da inteligência artificial e do processamento de linguagem natural. Esses modelos são sistemas de aprendizado profundo capazes de lidar com enormes quantidades de dados textuais, permitindo uma compreensão mais abrangente e contextualizada da linguagem humana. Ao contrário de seus predecessores, os LLMs são treinados em conjuntos de dados vastos e diversificados, extraindo padrões complexos e nuances da linguagem. Eles são construídos com milhões ou até bilhões de parâmetros, o que lhes confere uma capacidade excepcional de capturar e representar nuances semânticas, estruturas gramaticais e até mesmo contextos sutis em diferentes domínios linguísticos (BROWN *et al.*, 2020).

A arquitetura subjacente desses modelos geralmente se baseia em redes neurais profundas, como as *Transformers*, que têm a capacidade de processar sequências extensas de texto de forma eficiente (VASWANI *et al.*, 2017). Por meio de técnicas avançadas de aprendizado auto-supervisionado, esses modelos são treinados para prever palavras ou *tokens* em um texto com base no contexto fornecido, desenvolvendo uma compreensão profunda das relações e significados linguísticos. Deste modo, ao passar um texto como entrada para o modelo, este gera como saída um texto que tenha a maior probabilidade de completar o texto de entrada (RADFORD *et al.*, 2019).

O texto passado como entrada para o LLM é formalmente chamado de *prompt*. Um *prompt* em um LLM, representa uma entrada textual estruturada fornecida ao modelo para direcionar sua geração de texto ou resposta. Esta entrada serve como um estímulo ou guia para orientar o modelo na execução de tarefas específicas, podendo consistir em frases, palavras-chave, parágrafos ou qualquer forma de texto que direcione a compreensão e a saída do modelo, influenciando sua capacidade de processar informações e gerar respostas contextualmente relevantes dentro de um determinado domínio ou para uma tarefa específica.

Os principais exemplos de modelos deste tipo são BERT (DEVLIN *et al.*, 2018), GPT (BROWN *et al.*, 2020), LLaMa (TOUVRON *et al.*, 2023).

### 2.7.1 Abordagens de Passagem do KG para o LLM

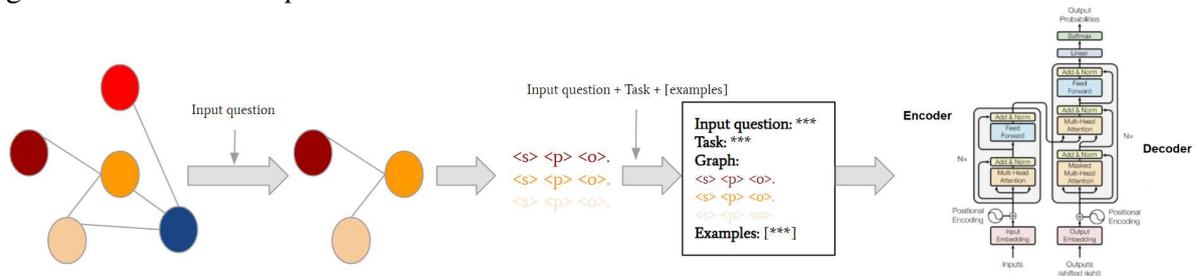
Nesta seção são apresentadas as principais abordagens para a integração do conhecimento contido no KG aos LLMs. As abordagens apresentadas nas Seções 2.7.1.1, 2.7.1.2 e 2.7.1.3 passam as informações diretamente através do *prompt* dado como entrada para o LLM. Por sua vez, a abordagem apresentada na Seção 2.7.1.4 utiliza técnicas de *knowledge graph embeddings* (Seção 2.6) para passar o conhecimento do KG para o LLM.

#### 2.7.1.1 Graph on Prompt

Nesta abordagem trechos do KG são passados diretamente ao LLM através do *prompt* de entrada. Desta maneira o LLM utiliza as triplas do KG como contexto para responder perguntas. O processo de passagem do grafo nesta abordagem pode ser visto de maneira resumida como se segue:

1. Seleciona fragmentos do grafo relevantes para a questão de entrada;
2. Gera uma serialização dos fragmentos selecionados;
3. Gera *prompt* de entrada para o LLM contendo a questão de entrada original, a serialização gerada do grafo e as instruções da tarefa a ser executada. Opcionalmente, o *prompt* pode conter poucos exemplos de execução da tarefa ou não.

Figura 4 – KG on Prompt.



Fonte: Próprio autor.

A Figura 4 ilustra o processo de ingestão do Grafo para o LLM como anteriormente descrito na abordagem *KG on Prompt*.

Como vantagens desta abordagem destacam-se:

- Simplicidade: Sendo a abordagem mais simples e direta, limitando-se a passar uma serialização direta como parte do *prompt* de entrada do LLM;
- Facilmente referenciável: Ao receber diretamente as triplas do KG, o LLM pode apontar de maneira simples e direta quais triplas foram utilizadas para embasar suas respostas e

conclusões;

- Amplo escopo de questões respondíveis: Esta abordagem permite que um amplo número de questões sejam respondíveis, seja através da análise direta das triplas passadas como entrada, ou através da escrita de consultas em linguagens formais a serem executadas sobre o KG por inteiro;
- Reuso da estrutura das triplas: Ao receber diretamente as triplas do KG, o LLM pode utilizar partes destas na escrita de consultas em linguagens formais de consultas, tais como SPARQL, desta maneira garantido que a consulta gerada siga de maneira fiel o formato em que os dados estão armazenados.

Como desvantagens desta abordagem destacam-se:

- Difícil interpretação: Esta abordagem depende de que o LLM utilizado seja capaz de compreender, interpretar e generalizar o formato de serialização passado como entrada. Além disso, o *prompt* gerado apresenta uma difícil interpretação para humanos;
- Alto consumo de espaço: Dependendo do tamanho do subgrafo selecionando (com o formato da serialização também impactando), as triplas podem ocupar um grande espaço na entrada, tornando a abordagem impraticável para grafos maiores;
- Tamanho de entrada limitado: LLMs possuem um tamanho de entrada máximo limitado pelo número de *tokens* do *prompt*;
- Impossibilidade da passagem de uma visão global do grafo: Por conta do tamanho máximo limitado do *prompt* de entrada, torna-se impossível para o LLM ter uma visão global do grafo, limitando-se a uma visão local presente do sub-grafo passado como entrada.

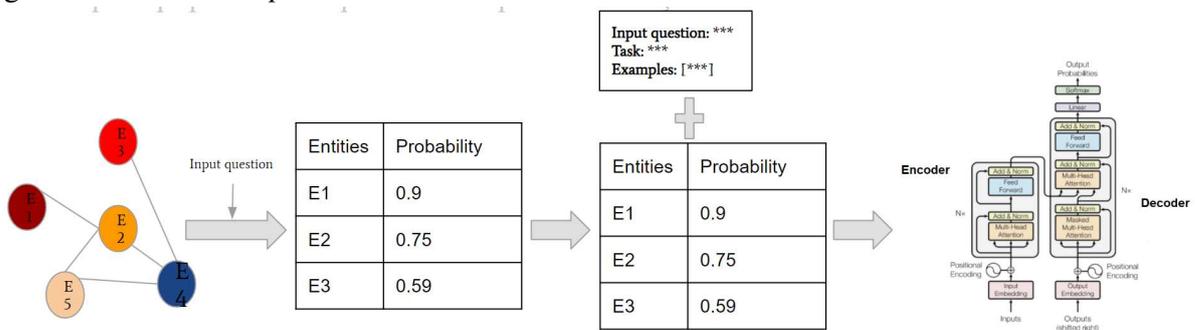
### 2.7.1.2 IR on Prompt

Nesta abordagem ferramentas de *Information Retrieval (IR)* são utilizadas para recuperar no grafo uma lista ranqueada de entidades relevantes, juntas de suas probabilidades de relevância para a questão de entrada. Esta abordagem pode ser utilizada em conjunto com a abordagem *KG on Prompt* para passar mais informações sobre as entidades selecionadas. No entanto, em sua abordagem pura, apenas a lista de recursos relevantes é passada, limitando o escopo de questões respondíveis. O processo de passagem do grafo nesta abordagem pode ser visto de maneira resumida como se segue:

1. Utiliza ferramentas de IR para selecionar entidades do grafo relevantes para a questão de entrada;

2. Gera *prompt* de entrada para o LLM contendo a questão de entrada original, as instruções da tarefa a ser executada, as entidades recuperadas acompanhadas de suas probabilidades de relevância. Opcionalmente o *prompt* pode conter poucos exemplos de execução da tarefa ou não.

Figura 5 – *IR on Prompt*.



Fonte: Próprio autor.

A Figura 5 ilustra o processo de ingestão do grafo para o LLM como anteriormente descrito na abordagem *IR on Prompt*.

Como vantagens desta abordagem destacam-se:

- Fácil implementação: Esta é possivelmente a abordagem com implementação mais simples, podendo limitar-se à chamada de ferramentas de *IR* já existentes e passado sua saída para o LLM;
- Indicação explícita de conceitos relevantes: Nesta abordagem as entidades relevantes acompanhadas de seus graus de relevância para a questão são fornecidas como entrada para o LLM, desta maneira dando dicas claras para o LLM sobre quais termos devem ser considerados na resposta final;

Como desvantagens desta abordagem destacam-se:

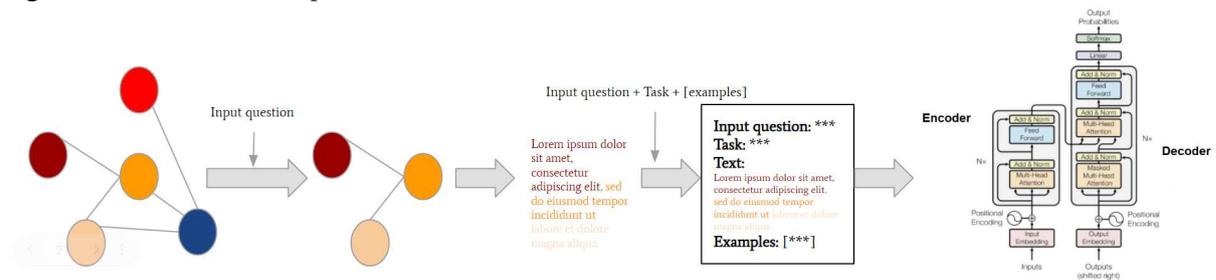
- Conhecimento mínimo sobre o grafo: Considerada de maneira independente, esta é abordagem onde é passada a menor quantidade de informação para o LLM, limitando-se à lista de entidades, ignorando os atributos e relacionamentos entre as entidades do grafo;
- Tamanho de entrada limitado: LLMs possuem um tamanho de entrada máximo limitado pelo número de *tokens* do *prompt*;
- Escopo limitado de questões respondíveis: Por considerar apenas entidades, o escopo de questões respondíveis limita-se à recuperação de entidades. Desta maneira o LLM atua como um refinamento do sistema de *IR* subjacente.

### 2.7.1.3 Text on Prompt

Nesta abordagem trechos do KG são verbalizados para então serem passados ao LLM através do *prompt* de entrada. A verbalização de triplas consiste no processo de traduzir triplas do KG para texto em linguagem natural que represente o mesmo significado semântico (BLINOV, 2020). O processo de passagem do grafo nesta abordagem pode ser visto de maneira resumida como se segue:

1. Seleciona fragmentos do grafo relevantes para a questão de entrada;
2. Gera uma verbalização dos fragmentos selecionados;
3. Gera *prompt* de entrada para o LLM contendo a questão de entrada original, a verbalização gerada do grafo e as instruções da tarefa a ser executada. Opcionalmente, o *prompt* pode conter poucos exemplos de execução da tarefa ou não.

Figura 6 – Text on Prompt.



Fonte: Próprio autor.

A Figura 6 ilustra o processo de ingestão do grafo para o LLM como anteriormente descrito na abordagem *Text on Prompt*.

Como vantagens desta abordagem destacam-se:

- Fácil interpretação: a verbalização do grafo é de fácil interpretação tanto para o LLM, quanto para usuários humanos, diminuindo o “*gap*” entre a questão de entrada e os dados consultados;
- Alta compatibilidade com LLMs: O processo de converter o conhecimento do KG em texto em linguagem natural possibilita que qualquer LLM disponível possa trabalhar diretamente com a entrada, independentemente se este foi treinado para interpretar triplas de um grafo.

Como desvantagens desta abordagem destacam-se:

- Qualidade de resposta dependente da verbalização: durante o processo de verbalização, conhecimento pode ser perdido ou traduzido incorretamente, desta maneira passando um contexto de menor qualidade para o LLM;

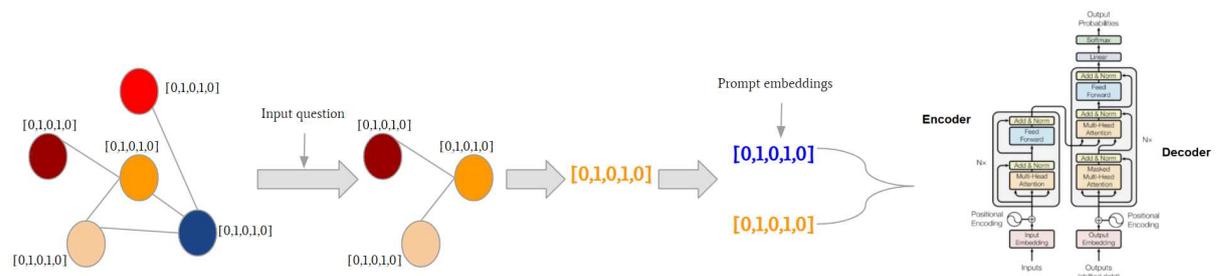
- Tamanho de entrada limitado: LLMs possuem um tamanho de entrada máximo limitado pelo número de *tokens* do *prompt*;
- Desconhecimento da estrutura interna do grafo: Ao receber apenas a verbalização, o LLM não tem acesso a informação estruturada de relacionamentos, identificadores de propriedades e recursos contidos no grafo. Desta maneira torna-se muito difícil a tarefa de construir consultas SPARQL que rodem na estrutura específica do KG;
- Impossibilidade da passagem de uma visão global do grafo: Por conta do tamanho máximo limitado do *prompt* de entrada, torna-se impossível para o LLM ter uma visão global do grafo, limitando-se a uma visão local presente do sub-grafo passado como entrada.

#### 2.7.1.4 KG embeddings

Nesta abordagem técnicas de *KG embeddings* (WANG *et al.*, 2017) são utilizadas para gerar representações vetoriais dos nós, arestas e/ou do próprio grafo por inteiro. Estes *embeddings* são então agregados aos *embeddings* da questão do usuário e passados diretamente para as camadas do LLM, dispensando a necessidade do uso de *prompts*. O processo de passagem do grafo nesta abordagem pode ser visto de maneira resumida como se segue:

1. Gera *embeddings* para os nós do grafo com o uso de *graph neural network* (GNN);
2. Seleciona o sub-grafo relevante para a questão de entrada (pode trabalhar também no nível de *embeddings*);
3. Gera *embeddings* no nível de grafo utilizando os *embedding* no nível de nós;
4. Gera *embeddings* do *prompt* de entrada contendo a questão de entrada e a tarefa a ser executada;
5. Agrega *embeddings* do *prompt* e do grafo para gerar *embeddings* de entrada;
6. Passa os *embeddings* diretamente para a camada pós geração de *embeddings* do *prompt* do LLM.

Figura 7 – KG Embeddings.



Fonte: Próprio autor.

A Figura 7 ilustra o processo de ingestão do Grafo para o LLM como anteriormente descrito na abordagem *KG embeddings*.

Como vantagens desta abordagem destacam-se:

- Não limitado ao limite de tamanho máximo de *tokens* no *prompt* de entrada: Por trabalhar diretamente como os vetores que serão passados para as camadas mais internas do LLM, estes podem codificar qualquer informação de qualquer tamanho para um vetor de tamanho fixo;
- Visão geral do grafo: Os *embeddings* gerados levam em consideração a estrutura e o relacionamento de todos os nós do KG original, desta maneira passando uma visão geral da estrutura do grafo para o LLM;
- Captura conhecimento não explícito para humanos: Os *embeddings* aprendidos da estrutura do grafo podem capturar padrões e conhecimentos não compreensíveis para humanos, mas que podem ser interpretados pelo LLM.

Como desvantagens desta abordagem destacam-se:

- Grande complexidade de implementação: Esta abordagem é aquela que apresenta a maior complexidade do ponto de vista de implementação, requerendo o uso de técnicas avançadas para geração dos *embeddings* e a manipulação direta do LLM, desprezando APIs que facilitem seu uso;
- Necessidade de recalcular os *embeddings* ao atualizar o grafo: Como os *embeddings* são aprendidos a partir da estrutura do grafo, ao inserir novos nós ou arestas, ou modificar os já existentes, os *embeddings* necessitam ser treinados novamente;
- Difícil interpretação por humanos: Por manipular apenas representações vetoriais, torna-se praticamente impossível a compreensão da entrada e do processo de raciocínio para chegar na resposta por parte de humanos.

## 2.8 In-context Learning

*In-context Learning* é a habilidade do modelo aprender temporariamente do *prompt* de entrada. Neste contexto, uma tarefa específica pode ser passada como entrada através do *prompt* de entrada, contendo instruções sobre o processo a ser executado na tarefa específica, bem como exemplos da execução desta tarefa. No caso de haver a passagem de poucos exemplos no *prompt*, esta estratégia é denominada *few-shot learning*. Por outro lado, nos casos onde não há a passagem de exemplos, esta estratégia é denominada *zero-shot learning*.

Neste contexto, a maneira como o *prompt* de entrada é estruturado é de vital importância para o aprendizado com eficiência da tarefa. Deste modo surge o estudo da engenharia de *prompts* (*Prompt Engineering*) para analisar o processo de estruturar texto que pode ser interpretado e entendido por um modelo generativo de IA na forma de um *prompt*.

A engenharia de *prompts* emergiu como um campo crucial na maximização do potencial de LLMs. Esta disciplina concentra-se na criação estratégica e otimização de entradas textuais para direcionar a geração de texto e as respostas dos modelos, influenciando sua capacidade de compreender contextos específicos e produzir saídas mais precisas e relevantes.

As estratégias de engenharia de *prompts* abrangem uma variedade de técnicas, desde a formulação cuidadosa de perguntas específicas até a seleção de palavras-chave e a formatação de instruções detalhadas. A manipulação precisa dessas entradas textuais visa orientar o LLM para contextos particulares, desbloqueando seu potencial máximo e permitindo a geração de respostas mais informadas e contextualizadas. Algumas das possíveis técnicas de engenharia de *prompt* são: *Chain-of-thought*, *Generated knowledge prompting*, *Least-to-most prompting*, *Self-refine*, *Tree-of-thought*, *Retrieval Augmented Generation (RAG)*, etc. (WIKIPEDIA, 2023).

## 2.9 Model fine-tuning

O ajuste fino do modelo (*model fine-tuning*) é uma abordagem comum, que consiste na adaptação do modelo pré-treinado para uma tarefa específica, utilizando um conjunto de dados anotados relativos à tarefa desejada.

Por conta do grande número de parâmetros presentes em LLMs, o processo de re-treinamento do modelo demanda uma grande quantidade de recursos de memória, processamento e tempo, tornando-se impraticável para pequenas organizações. Deste modo surgiu o campo de pesquisa de *Parameter Efficient Fine-tuning* (PEFT).

O PEFT concentra-se na identificação e otimização de um conjunto central de parâmetros essenciais dentro de LLMs durante o processo de ajuste fino. Ao invés de ajustar todos os parâmetros do modelo, o PEFT identifica e prioriza especificamente aqueles que têm maior impacto na tarefa em questão, permitindo um ajuste mais direcionado e eficiente.

O método de PEFT mais popular no ajuste de LLMs é o método *Low-Rank Adaptation* (LoRA) (HU *et al.*, 2021). A abordagem LoRA se baseia na decomposição de matrizes de parâmetros em estruturas de baixa dimensionalidade, como fatores latentes ou subespaços de menor rank. Em vez de ajustar diretamente todos os parâmetros do LLM, o LoRA congela os

pesos pré-treinados do modelo e insere matrizes de decomposição de baixo *rank* treináveis em cada camada da arquitetura *Transformer*, reduzindo significativamente o número de parâmetros treináveis para tarefas posteriores. Comparado ao GPT-3 175B ajustado com o método Adam, o LoRA pode reduzir o número de parâmetros treináveis em 10.000 vezes e os requisitos de memória da GPU em 3 vezes (HU *et al.*, 2021).

Tendo em vista a diminuição no uso de memória na abordagem LoRA, a abordagem QLoRA utiliza técnicas de quantização para aumentar a eficiência e diminuir o espaço ocupado no treinamento de LLMs (DETTMERS *et al.*, 2023).

Uma das principais vantagens do ajuste fino de LLMs é a capacidade de adaptar-se a domínios ou tarefas específicas com uma quantidade relativamente menor de dados de treinamento. Ao passo que os modelos pré-treinados possuem uma compreensão ampla da linguagem, o *fine-tuning* permite que se ajustem de maneira mais focada para contextos especializados, aprimorando assim sua capacidade de fornecer respostas mais precisas, atualizadas e relevantes em cenários específicos.

## 2.10 ChatGPT

ChatGPT<sup>6</sup> é um modelo de linguagem desenvolvido pela OpenAI, projetado para interagir e gerar respostas similares às humanas em linguagem natural. Baseado na arquitetura de redes neurais conhecida como GPT (*Generative Pre-trained Transformer*), o ChatGPT é treinado em uma ampla gama de texto da internet, permitindo-lhe compreender e produzir respostas em diversos domínios.

Dentre os domínios aos quais o ChatGPT destaca-se, encontra-se sua capacidade de interpretar grafos RDF, bem como escrever consultas SPARQL sobre os mesmos. Deste modo, neste trabalho foi investigada a capacidade do ChatGPT atuar como uma ferramenta de KGQA.

Embora o ChatGPT seja hoje uma das ferramentas mais avançadas no que tange a LLMs, seu uso está condicionado a algumas limitações, tais como:

- Restrições para a realização de *fine-tuning* do modelo, estando disponível apenas para certos modelos em alguns planos de contratação dos serviços;
- Impossibilidade de passagem direta de *embeddings* para o modelo;
- Limite no número de *tokens* utilizados, contando entrada e saída.

---

<sup>6</sup> <https://chat.openai.com/>

## 2.11 RDF Browsers

*RDF Browser* ou *Semantic Web Browser* são aplicações que localizam automaticamente metadados e montam interfaces de apontar e clicar a partir de especificações ontológicas descritas em RDF e recuperadas dinamicamente da Web Semântica (QUAN; KARGER, 2004). Deste modo, tais ferramentas podem ser reutilizadas no contexto de KGs armazenados no formato RDF.

Atualmente existem dois tipos de RDF Browsers, os *text-based* (**baseados em textos**), tais como o *OpenLink Faceted Browser*<sup>7</sup>; bem como os *visual-based* (**baseados em visual**), tais como o *LodLive*<sup>8</sup>. Os *Browsers* do primeiro tipo tendem a ser mais simples e leves, apresentando os dados como uma simples página HTML contendo listas de valores para propriedades do recurso RDF sendo explorado. Por sua vez, os do segundo tipo tendem a ser mais complexos e com mais recursos, apresentando o grafo em uma representação visual composta por nós e arestas. As interfaces baseadas em visual são o tipo que mais vem sendo explorado e recebendo novos recursos recentemente.

---

<sup>7</sup> <https://github.com/OpenLinkSoftware/fct>

<sup>8</sup> <http://en.lodlive.it/>

### 3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados os trabalhos relacionados encontrados na literatura. Na Seção 3.1 são apresentadas as abordagens para seleção de sub-grafos relevantes para a pergunta relacionadas à abordagem seguida nesta tese. A Seção 3.2 apresenta as principais abordagens de KGQA baseadas em LLMs. Por fim, a Seção 3.3 apresenta os principais *datasets* utilizados pelas abordagens apresentadas na Seção 3.2.

#### 3.1 Abordagens para seleção do sub-grafo relevante para perguntas

ATHENA é um sistema de QA baseado em ontologias para banco de dados relacionais complexos (SAHA *et al.*, 2016) com um sistema embutido para seleção automática do sub-grafo relevante para a pergunta em linguagem natural. A abordagem empregada por ATHENA mapeia os *tokens* da entrada para termos contidos na ontologia. O sistema utiliza mapeamentos entre listas de sinônimos e elementos da ontologia para relacionar a entrada com o vocabulário do grafo. Estes sinônimos podem ser inseridos manualmente ou recuperados automaticamente de recursos como a WordNet (MILLER, 1995). Este processo é então completado pela geração de árvores de interpretação da consulta conectando os candidatos aos termos da ontologia por meio da execução de um algoritmo de *steiner-tree*, construindo árvores de interpretação da consulta com base em um conjunto de heurísticas definidas pelos autores. A melhor árvore de interpretação é escolhida com base em um conjunto de evidências, também definidas pelos autores, dando origem a uma consulta intermediária. Em seguida, a consulta intermediária é traduzida para SQL sobre o banco relacional original. Do ponto de vista de seleção do sub-grafo relevante para a pergunta, a abordagem adotada por ATHENA é limitada ao uso de associações exatas entre termos da pergunta e sinônimos de termos da ontologia, onde estes podem não cobrir completamente todas as possibilidades existentes, tornando a abordagem menos generalizável. Além disso, o processo de expansão do sub-grafo selecionado está intimamente preso ao processo de interpretação da pergunta com base em heurísticas definidas manualmente. Por conta da forte dependência de heurísticas e métricas definidas manualmente pelos autores, esta abordagem possui um escopo limitado de questões respondíveis, limitando seu uso prático em cenários não controlados.

Em (TIAN *et al.*, 2023), os autores propõem *Graph Neural Prompting* (GNP), um método *plug-and-play* para auxiliar LLMs pré-treinados a aprender conhecimento benéfico

de KGs. GNP codifica o conhecimento presente no grafo como um vetor de *embeddings* que pode ser enviado para LLMs para prover orientação e instruções. GNP utiliza uma *graph neural network* (GNN) para capturar e codificar o grafo em *embeddings* de entidades e nós. Em seguida, um módulo de *cross-modality pooling* (agrupamento intermodal) é utilizado para determinar os *embeddings* mais relevantes para o texto de entrada, consolidando-os em um *embedding* holístico de nós em nível de grafo. Após isto, GNP utiliza um *domain projector* (projektor de domínio) para tratar disparidades inerentes entre o grafo e os textos do domínio. Por fim, uma função objetivo de predição de links auto-supervisionada (*self-supervised link prediction objective*) é utilizada para aprimorar a compreensão do modelo sobre relacionamentos entre entidades e capturar o conhecimento do grafo em uma maneira auto-supervisionada. A abordagem proposta foi avaliada em múltiplos *datasets* de *benchmarks* públicos nos domínios de raciocínio de conhecimento de senso comum e biomédico, alcançando uma melhoria de 13.5% ao comparar com LLMs pré-treinados e uma melhoria de 1.8% ao comparar com LLMs ajustados utilizando apenas a abordagem *LoRA*. A abordagem apresentada depende do acesso direto à estrutura interna do LLM para realizar o treinamento do modelo e passagem dos *embeddings* como entrada, o que pode limitar o seu uso em LLMs comerciais de terceiros, tais como o ChatGPT da OpenAI e o Gemini do Google (Google, 2024).

Em (FENG *et al.*, 2023), os autores propõem o paradigma *Knowledge Solver* (KSL), para ensinar LLMs a buscarem conhecimento essencial de bases de conhecimento externas. Nesta abordagem o LLM recebe como entrada pares de questões e respostas (do tipo múltipla escolha), onde a tarefa do LLM é aprender a selecionar o subgrafo do KG necessário para responder a questão original. Para isto, os autores desenvolveram uma estratégia de *prompt* para transformar recuperações em uma sequência de decisões *multi-hop*. Para isto, em cada etapa, a informação do KG é transformada em *prompts* de texto (incluindo o caminho histórico escolhido pelo LLM), baseado no que o LLM considera relevante no contexto para executar a tarefa. Na abordagem descrita, inicialmente são identificadas as entidades presentes na questão e no conjunto de resposta, onde o LLM deve escolher quais relacionamentos devem ser selecionados para alcançar a entidade final da resposta em uma maneira iterativa. A abordagem foi testada em três diferentes LLMs (GPT3.5, LLaMa e LLaMa2) em três diferentes *datasets* (*CommonsenseQA*, *OpenbookQA*, and *MedQA-USMLE*), alcançando uma sensível melhoria de acurácia nos dois primeiros *datasets* em uma configuração *zero-shot*, alcançando resultados ainda melhores ao realizar o *fine-tuning* do modelo. A abordagem apresentada depende de um grande número de

chamadas ao LLM para selecionar o sub-grafo relevante para a questão, o que pode acarretar em grandes custos computacionais e monetários em cenários utilizados LLMs proprietários de terceiros.

A principal diferença do mecanismo de seleção do sub-grafo implementado nesta tese com relação aos apresentados em (TIAN *et al.*, 2023) e (FENG *et al.*, 2023) é que estes dois dependem da realização de chamadas ao LLM nesta etapa, o que pode aumentar consideravelmente o custo (computacionalmente e monetariamente) desta operação. Ao comparar à abordagem proposta no sistema ATHENA (SAHA *et al.*, 2016), a abordagem proposta nesta tese utiliza métricas baseadas nas distâncias dos vetores de *embeddings* dos termos do KG, alcançando uma maior generalização, não dependendo de listas de sinônimos fixas, nem de heurísticas definidas manualmente.

### 3.2 Abordagens de KGQA baseadas em LLMs

Em (RONY *et al.*, 2022), os autores propõem SGPT, uma abordagem para tradução de consultas em linguagem natural para SPARQL. SPGPT codifica para um vetor de *embeddings* *features* linguísticas, tais como, parts-of-speech (POS) e a árvore de dependência, além do sub-grafo correspondente da questão. Em seguida os *embeddings* são passados como treinamento para um modelo de *Transformers* baseado no GPT-2 para gerar a consulta SPARQL. O modelo é avaliado nos *datasets* LC-QuAD 2.0, VQuAnDA e QALD-9, sendo até hoje o estado da arte neste último. A abordagem apresentada tem como limitação a necessidade do recebimento do sub-grafo correspondente da questão como entrada, além de necessitar do acesso direto à estrutura interna do LLM para realizar o treinamento do modelo e passagem dos *embeddings* como entrada.

Em (QI *et al.*, 2023), os autores apresentam *FoodGPT*, um LLM para o domínio de testes de comida. Para a construção de *FoodGPT*, os autores utilizaram fontes de dados estruturadas e documentos escaneados para o treinamento incremental da rede, além do uso de um KG externo como fonte de consulta para reduzir o fenômeno de alucinação. Para imagens e documentos escaneados os autores utilizaram técnicas de *optical character recognition* (OCR) para extrair informações textuais das fontes originais. Em seguida são utilizados os modelos GPT-2 e BERT para remover ruídos e textos com baixa qualidade, que em seguida serão utilizados como corpus de treinamento para o LLM, acrescidos de prefixos que indiquem sua proveniência. Para dados estruturados, os autores utilizam extração de dados baseados em regras (*templates*),

assim como geração de texto utilizando o *ChatGPT* para gerar texto em linguagem natural que represente os dados estruturados. Em seguida, o modelo *Chinese-LLaMA2-13B* e o método *LoRA* foram utilizados para a etapa de *fine-tuning* do modelo. Para a utilização do KG, *FoodGPT* utiliza um *RAG* para analisar a consulta do usuário e recuperar conhecimento relevante do KG. O conhecimento recuperado do KG é então concatenado com a consulta e passado como entrada para o *FoodGPT*, gerando então a resposta com base no fragmento do grafo dado e em seu conhecimento adquirido durante o treinamento. Esta abordagem apresenta um modelo rudimentar de integração do conhecimento do KG ao LLM, recorrendo ao uso de um mecanismo padrão de *RAG*. Os experimentos apresentados na Seção 6.1 indicam que tais mecanismos não são capazes de selecionar adequadamente todas as informações do KG relevantes para a pergunta.

Em (WU *et al.*, 2023), os autores propõem uma abordagem sensível à resposta para transformação de KGs em texto, gerando uma representação “bem-textualizada” das declarações contidas no KG. Com base nesta abordagem, os autores propõem um *framework* para a tarefa de KGQA, baseada em LLMs aumentados por *KG-to-Text*. O *framework* é composto por três etapas: recuperação de sub-grafo, transformação do sub-grafo para texto (*KG-to-Text*) e raciocínio com conhecimento de texto aprimorado. O processo de seleção do sub-grafo envolve três etapas principais: predição de saltos, predição de caminho de relação e amostragem de triplas. Primeiro, a predição de saltos estima o número de saltos necessários para responder a uma pergunta, modelada como uma tarefa de classificação usando modelos de linguagem pré-treinados (PLMs). A pergunta é codificada em uma representação vetorial, que é então usada para prever a distribuição de probabilidade de números de saltos potenciais. O número de saltos com a maior probabilidade é, então, selecionado. Em seguida, na predição de caminho de relações, dada a pergunta e o número previsto de saltos, os caminhos de relações mais prováveis são identificados por meio de um processo de predição passo a passo. Os caminhos de relações são pontuados com base no produto de suas probabilidades individuais. Finalmente, a amostragem de triplas recupera triplas relevantes do KG com base nos caminhos de relações previstos para construir o sub-grafo necessário para responder à pergunta. Testes em vários *datasets* apontaram que a abordagem supera abordagens anteriores baseadas em LLMs aumentadas por KGs. Para a transformação *KG-to-Text*, os autores implementaram um LLM treinado (*fine-tuned*) para este propósito, onde para tratar a escassez de corpus *KG-to-Text*, os autores projetaram um método de geração automática de corpus *KG-to-Text*, utilizando o *ChatGPT*, diferentemente de métodos

anteriores que utilizaram GNNs.

Em (LUO *et al.*, 2023), os autores apresentam CHATKBQA, um *framework* para realizar *fine-tuning* em LLMs *open-source* para *generate-then-retrieve* KGQA. Para isto, o *framework* primeiramente gera uma forma lógica da consulta dada como entrada, em seguida, realizando a etapa de recuperação de entidades e relacionamentos. Neste processo, CHATKBQA busca separar as duas etapas, tendo como objetivo evitar a influência da recuperação na geração da forma lógica e aprimorar a eficiência da recuperação. Para a geração da forma lógica, CHATKBQA utiliza LLMs *open-source* treinados utilizando técnicas de *instruction tuning* (OUYANG *et al.*, 2022), utilizando máscaras no lugar de entidades e relações. Para a fase de recuperação de entidades e relacionamentos, a abordagem utiliza um método de recuperação não supervisionado que conduz a recuperação semântica em nível de frases no conjunto de entidades e relacionamentos. Ao avaliar o *framework* nos *datasets* *WebQSP* e *ComplexWebQuestions*, o mesmo conseguiu atingir uma nova performance de estado da arte na tarefa de KGQA. A abordagem apresentada depende do acesso direto à estrutura interna do LLM para realizar o treinamento do modelo.

Em (LEHMANN *et al.*, 2023), os autores propõem o uso de linguagem natural controlada como alvo na tarefa de *semantic parsing* para KGQA, ao invés do uso de linguagens lógicas formais, tais como SPARQL. Para avaliar sua hipótese, os autores desenvolveram um novo *dataset*, construído manualmente, baseado no *dataset* Mintaka (SEN *et al.*, 2022), contendo 550 pares de questões, suas formas lógicas correspondentes e suas respostas. O *dataset* gerado conta com as formas lógicas SPARQL, Sparklis e SQUALL, onde as duas últimas são consultas em linguagem natural controlada. A abordagem proposta primeiramente realiza o *fine-tuning* na tarefa de tradução da questão de entrada para linguagem alvo desejada. Nos experimentos apresentados pelos autores os seguintes LLMs foram utilizados BLOOM, GPT Neo2, GPT-2, GPT-3, T5 e Llama 2, com o GPT-3 alcançando os melhores resultados nos cenários testados. Durante o tempo de execução, a abordagem proposta inicialmente utiliza o *Entity Linker ReFinED* (AYOOLA *et al.*, 2022) para encontrar a lista de entidades contidas na questão original, em seguida realizando a recuperação de informações acerca destas. Em seguida, a concatenação da questão, entidades e suas probabilidades e informações encontradas para cada uma são usadas como entrada para a LLM que irá gerar uma consulta intermediária na linguagem alvo que em seguida passará por uma etapa de ajustamento de relações, gerando então a consulta final. A abordagem apresentada depende de *fine-tuning* do LLM para a tarefa de tradução da pergunta

para a linguagem de consulta alvo, assim dependendo da qualidade do *dataset* disponível para esta tarefa, tendo ocorrido a necessidade da construção manual deste.

Em (KOVRIQUINA *et al.*, 2023), os autores apresentam SPARQLGEN, uma abordagem generativa *one-shot* para gerar consultas SPARQL por aumentar LLMs com contexto relevante dentro de um único *prompt*. O *prompt* proposto é composto por (1) a questão a ser traduzida (2) um subgrafo RDF requerido para traduzir a consulta, (3) um exemplo de uma consulta SPARQL correta para uma questão diferente. A abordagem utilizou o GPT-3 para testar em 3 diferentes datasets QALD-9, QALD-10 e BESTIARY (*dataset* construído pelos próprios autores). Na abordagem utilizada, a seleção dos sub-grafos passados como entrada no *prompt* para cada questão foi realizada utilizando um processo de engenharia reversa nas consultas SPARQL corretas esperadas (*ground truth query*), construído o sub-grafo a partir dos *basic graph patterns* (BGP)<sup>1</sup> presentes na consulta, substituindo variáveis pelos seus valores retornados ao executar a consulta SPARQL projetando todas as variáveis envolvidas, desta maneira gerando o sub-grafo mínimo para responder a consulta. Como resultado, os autores descobriram que ao passar o KG subjacente e um exemplo de consulta aleatória melhora os resultados de geração. A dependência de engenharia reversa para selecionar sub-grafos específicos limita sua escalabilidade e generalização.

Sequeda et al. (SEQUEDA *et al.*, 2023) introduziu um *benchmark* que compreende um esquema SQL corporativo no domínio de seguros, uma série de consultas corporativas, e uma camada contextual incorporando uma ontologia e mapeamentos que definem um KG como uma visão semântica do banco relacional. Os autores mostraram que responder perguntas usando GPT-4, com *prompts* zero-shot diretamente em bancos de dados SQL, alcançou uma precisão de 16%, e aumentou para 54% quando as perguntas foram feitas sobre o KG representando o banco de dados SQL corporativo. A Seção 6.2 indica que o Auto-KGQA melhora esses resultados enquanto reduz o número de *tokens* passados para o LLM. A principal melhoria desta tese consiste na passagem de informações tanto do T-Box quanto do A-Box para o LLM, combinada com o uso de um algoritmo de filtragem. O mecanismo de filtragem gerou uma redução de 51~87% no número de *tokens* do KG, viabilizando a passagem de triplas do A-Box que atuam como exemplos reais de como os dados são representados.

Em (LEHMANN *et al.*, 2024) os autores apresentam uma série de experimentos para avaliar o desempenho de diversos modelos de LLMs sobre as questões do *benchmark*

<sup>1</sup> [https://www.w3.org/2001/sw/DataAccess/rq23/sparql-defns.html#defn\\_BasicGraphPattern](https://www.w3.org/2001/sw/DataAccess/rq23/sparql-defns.html#defn_BasicGraphPattern)

SciQA. Os experimentos buscaram avaliar o impacto de técnicas tais como *fine-tuning* e *few-shot learning* no desempenho dos modelos. Dentre os modelos avaliados encontram-se os modelos T-5, Dolly, GPT 2 e GPT 3.5 Turbo, onde nos dois primeiros foram avaliadas técnicas de *fine-tuning*, enquanto técnicas de *few-shot learning* foram aplicadas em todos, exceto no primeiro. O GPT 2 foi o único modelo que passou por experimentos envolvendo *fine-tuning* e *few-shot learning*. De acordo com os resultados, a melhor abordagem foi utilizando o modelo T-5 com técnicas de *fine-tuning*, alcançando uma F-1 score de 0.9751. Ao considerar modelos sem o uso de *fine-tuning* os resultados da F-1 score ficaram abaixo de 0.30, alcançando 0.1087, 0.0653, e 0.2632 para os modelos Dolly, GPT 2 e GPT 3.5 Turbo respectivamente indicando que os modelos não possuem conhecimento sobre o modelo do KG sendo consultado. Ao considerar os melhores resultados utilizando a técnica de Few-shot learning (No máximo 7 amostras), revelou-se que o modelo GPT 2 não mostrou a capacidade de aprender via *in-context learning*, alcançando uma F-1 máxima de 0.3107. Por sua vez, Dolly e GPT 3.5 Turbo demonstraram ótimas capacidades para *in-context learning*, com o primeiro alcançando uma F-1 máxima de 0.8792 (1-shot learning) e o segundo com 0.9736 (7-shot learning). Os experimentos apresentados por (LEHMANN *et al.*, 2024) deixam claro que LLMs podem alcançar excelentes resultados quando existem exemplos que podem ser utilizados como contexto para os modelos. Tais exemplos pode ser utilizados em técnicas de *fine-tuning* ou *in-context learning*. No entanto, em casos onde tais exemplos não existem, tais modelos não alcançam desempenhos aceitáveis. Em cenários do mundo real a existência de exemplos não é garantida, portando o uso de técnicas como as aplicadas no framework desta tese, onde triplas selecionadas diretamente do KG são utilizadas como contexto para o LLM, mostram-se valiosas para sistemas de KGQA.

A Tabela 1 apresenta um resumo dos trabalhos relacionados avaliados. Nesta tabela torna-se visível que, ao melhor de meu conhecimento, excetuando (SAHA *et al.*, 2016), que todos os trabalhos relacionados quando não recebem o sub-grafo relevante para a questão como entrada da abordagem, necessitam de *fine-tuning* do LLM em algum estágio do processo. Deste modo, ao melhor de meu conhecimento, o *framework* proposto nesta tese é o primeiro sistema de KGQA baseado em LLMs capaz de executar a seleção do sub-grafo relevante para questão de maneira autônoma e sem a necessidade da realização de *fine-tuning*.

Tabela 1 – Consolidação dos dados extraídos dos trabalhos estudados.

Trabalho	Necessita fine-tuning	Traduz consulta	Seleciona contexto automaticamente	Datasets	LLM
(RONY <i>et al.</i> , 2022)	Sim	Sim	Não	LC-QuAD 2.0, VQuAnDA, QALD-9	GPT-2
(QI <i>et al.</i> , 2023)	Sim	Não	Sim	Dataset próprio	LLaMA2
(WU <i>et al.</i> , 2023)	Sim*	Não	Sim	MetaQA, WebQSP, WebQ, ZhejiangQA (ZJQA)	Llama-2, T5, Flan-T5, T0
(LUO <i>et al.</i> , 2023)	Sim	Sim	Sim	WebQSP e Complex WebQuestions	Llama-2, ChatGLM2, Baichuan2
(LEHMANN <i>et al.</i> , 2023)	Sim	Sim	Sim	Dataset próprio (Mintaka-CNL)	BLOOM, GPT Neo2, GPT-2, GPT-3, T5, Llama 2
(KOVIRIGUINA <i>et al.</i> , 2023)	Não	Sim	Não	QALD-9, QALD-10, Dataset próprio: BESTIARY	GPT3
(SEQUEDA <i>et al.</i> , 2023)	Não	Sim	Não	Dataset próprio: Sequeda	GPT4
(LEHMANN <i>et al.</i> , 2024)	Sim	Sim	Não	SciQA	T5, Dolly, GPT 2, GPT 3.5
Este trabalho	Não	Sim	Sim	Dataset próprio, Sequeda, SciQA	GPT3.5, GPT-4 e GPT-4o

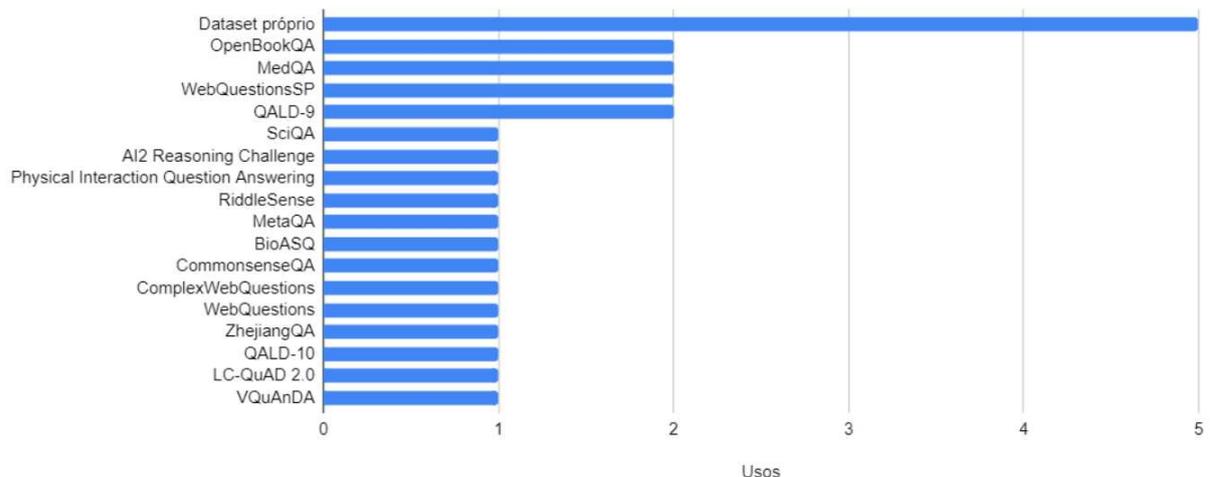
Fonte: elaborada pelo autor.

Legenda: \* *Fine-tuning* obrigatório apenas para a etapa de KG-to-Text.

### 3.3 Dataset e métricas utilizadas para avaliar sistemas de KGQA baseados em LLMs

Ao tentar identificar *datasets* padrões para a avaliação de sistemas de KGQA baseados em LLMs é possível notar que uma grande variedade de *datasets* são adotados por estes sistemas, não sendo possível identificar um *benchmark* padrão. A Figura 8 apresenta a contagem de trabalhos utilizando cada *dataset*, valendo destacar que um mesmo trabalho pode utilizar mais de um destes. Ao analisar a Figura 8 é possível notar que apenas os *datasets* *OpenBookQA*, *MedQA*, *WebQuestionsSP* e *QALD-9* são utilizados por 2 trabalhos, enquanto os demais são utilizados por apenas 1, sendo estes: *AI2 Reasoning Challenge*, *Physical Interaction Question Answering*, *RiddleSense*, *MetaQA*, *BioASQ*, *CommonsenseQA*, *ComplexWebQuestions*, *WebQuestions*, *ZhejiangQA*, *QALD-10*, *LC-QuAD 2.0*, *VQuAnDA*, *BESTIARY*. Vale ressaltar que 4 trabalhos utilizam *datasets* próprios, sendo a opção mais popular, no entanto, estes são *datasets* diferentes, embora contados na mesma linha.

Figura 8 – Quantidade de trabalhos utilizando cada dataset



Fonte: Próprio autor.

O *OpenBookQA* é um *dataset* de perguntas e respostas modelado com base em livros abertos de exames para avaliar a compreensão humana de um assunto. Ele consiste em 5.957 questões de múltipla escolha de nível elementar de ciências (4.957 de treinamento, 500 de desenvolvimento, 500 de teste), que investigam a compreensão de um pequeno "livro" de 1.326 fatos científicos essenciais e a aplicação desses fatos a novas situações. Para treinamento, o conjunto de dados inclui um mapeamento de cada questão para o fato científico essencial que foi projetado para investigar. Responder às perguntas do *OpenBookQA* requer conhecimento comum amplo adicional, não contido no livro. As perguntas, por *design*, são respondidas incorretamente por um algoritmo baseado em recuperação e um algoritmo de coocorrência de palavras. Redes

neurais fortes alcançaram cerca de 50% no OpenBookQA, deixando uma grande lacuna para a precisão de 92% dos trabalhadores de *crowd-work*. Por se tratarem de perguntas de múltipla escolha, este *dataset* não se encaixa no tipo de perguntas tratadas nesta tese. Além disso, o *dataset* não acompanha um KG que pode ser utilizado para responder as questões, ficando a cargo de cada abordagem propor o seu.

MedQA é um *dataset* de Resposta a perguntas de múltipla escolha com base nos Exames de Licença Médica dos Estados Unidos (USMLE). O conjunto de dados é coletado dos exames do conselho médico profissional. Ele abrange três idiomas: inglês, chinês simplificado e chinês tradicional, e contém 12.723, 34.251 e 14.123 perguntas para os três idiomas, respectivamente (JIN *et al.*, 2021). Por se tratarem de perguntas de múltipla escolha, este *dataset* não se encaixa no tipo de perguntas tratadas nesta tese. Além disso, o *dataset* não acompanha um KG que pode ser utilizado para responder as questões, ficando a cargo de cada abordagem propor o seu.

O *dataset* WebQuestionsSP contém parses semânticos completos em consultas SPARQL para 4.737 perguntas e anotações “parciais” para as 1.073 perguntas restantes para as quais uma análise válida não pôde ser formulada ou onde a pergunta em si é ruim ou precisa de uma resposta descritiva. As perguntas do *dataset* são formuladas sobre a *freebase* (BOLLACKER *et al.*, 2008).

Os desafios *Question Answering over Linked Data* (QALD) visam fornecer referências atualizadas para avaliar e comparar sistemas de última geração que mediam entre um usuário, expressando sua necessidade de informação em linguagem natural e dados RDF (NGOMO, 2018). O *dataset* conta com perguntas feitas em linguagem natural, suas respectivas traduções para consultas SPARQL (em duas versões, uma sobre a DBpedia e outra sobre a Wikidata) e seu conjunto de recursos de resposta.

O SciQA inclui 2.565 pares, cada um composto por uma pergunta em linguagem natural e a consulta SPARQL equivalente. As consultas SPARQL são projetadas especificamente para extrair informações do *Open Research Knowledge Graph* (ORKG). Este KG atualmente consiste em 170.000 recursos que detalham as contribuições de pesquisa de quase 15.000 artigos acadêmicos abrangendo 709 campos de pesquisa. O SciQA é uma mistura de pares de perguntas e consultas criados manualmente e gerados automaticamente. Especificamente, 100 pares foram criados manualmente por uma equipe de pesquisadores de diferentes países e instituições. Em seguida, a partir desses 100 pares iniciais, os autores identificaram oito modelos de consulta e os

usaram para produzir automaticamente 2.465 pares adicionais usando o modelo GPT-3 (AUER *et al.*, 2023).

A avaliação de sistemas de QA é fundamental para medir a eficácia e a precisão dessas ferramentas na compreensão e resposta a perguntas formuladas pelos usuários. Entre as métricas mais comumente adotadas para esse fim estão a precisão (*precision*), o revocação (*recall*) e o f1-score.

- **Precisão (*Precision*):** No campo da recuperação da informação, a precisão é a fração de documentos recuperados que são relevantes para a consulta. No contexto de QA são contados os recursos (objetos, instâncias ou valores de propriedades), com precisão sendo definida da seguinte forma:

$$\text{Prec}(Q) = \frac{\text{Número de recursos corretos retornados}}{\text{Número total de recursos retornados}} \quad (3.1)$$

- **Revocação (*Recall*):** No campo da recuperação da informação, o *recall* é a fração dos documentos relevantes que são recuperados com sucesso. No contexto de QA são contados os recursos (objetos, instâncias ou valores de propriedades), com *recall* sendo definido da seguinte forma:

$$\text{Rec}(Q) = \frac{\text{Número de recursos corretos retornados}}{\text{Número total de recursos corretos esperados}} \quad (3.2)$$

- ***F-measure (F1-score)*:** Média harmônica da precisão e do *recall*. Esta métrica dá um maior peso para valores baixos, deste modo, caso a precisão esteja muito elevada, mas o *recall* esteja muito baixa (e vice-versa), o valor da *F-measure* será menor do que a média simples esperada (LEARN, 2019). A *F-measure* é definida da seguinte forma:

$$\text{F-score}(Q) = 2 * \frac{\text{Prec}(Q) * \text{Rec}(Q)}{\text{Prec}(Q) + \text{Rec}(Q)} \quad (3.3)$$

- **Acurácia (*accuracy*):** A acurácia fornece uma medida direta da eficácia do sistema em fornecer respostas corretas, sendo uma métrica intuitiva e de fácil interpretação para a avaliação de desempenho em sistemas de QA. Uma acurácia alta indica que o sistema é capaz de responder corretamente à maioria das perguntas, enquanto uma acurácia baixa sugere que há espaço para melhorias no sistema. A acurácia é definida da seguinte forma:

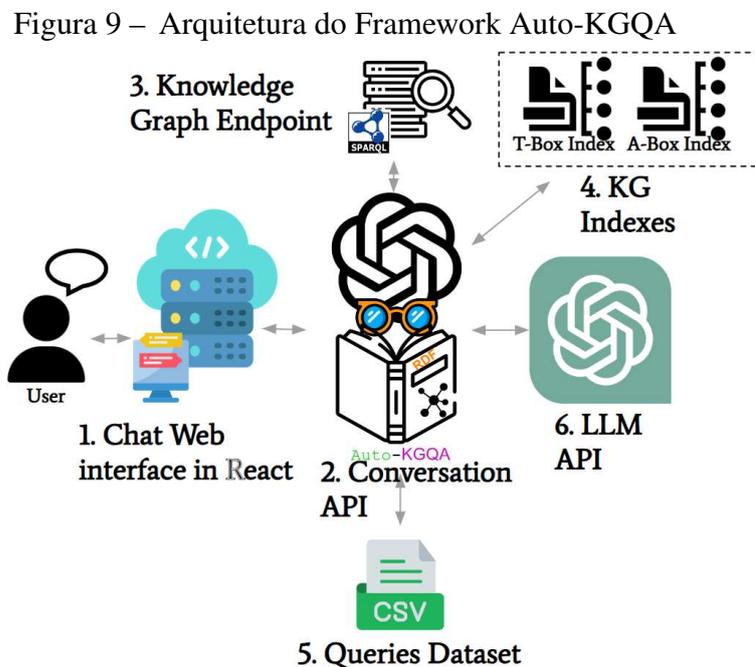
$$\text{Accuracy}(Q) = \frac{\text{Número de respostas totalmente corretas retornadas}}{\text{Número total de perguntas realizadas}} \quad (3.4)$$

#### 4 AUTO-KGQA: UM FRAMEWORK BASEADO EM LLM PARA KGQA

Nesta tese é proposto Auto-KGQA, um *framework* autônomo independente de domínio baseado no ChatGPT para a tarefa de KGQA. A discussão a seguir usa ChatGPT como LLM, mas o *framework* é genérico. A arquitetura do *framework* é apresentada na Figura 9, sendo composta por seis componentes:

1. *Chat Web Interface*, uma interface web de *instant messenger* para enviar e receber mensagens ao sistema;
2. *Conversation API - Auto-KGQA*, um sistema de de processamento de questões acessível via API HTTP;
3. *Knowledge Graph Endpoint*;
4. *KG indexes*, índices utilizados para encontrar referências aos recursos do KG em questões em linguagem natural;
5. *Dataset* de exemplo de consultas colhidas durante o uso;
6. API de LLM (ChatGPT por padrão).

O código do *framework* encontra-se publicamente disponível no repositório do github<sup>1</sup>.



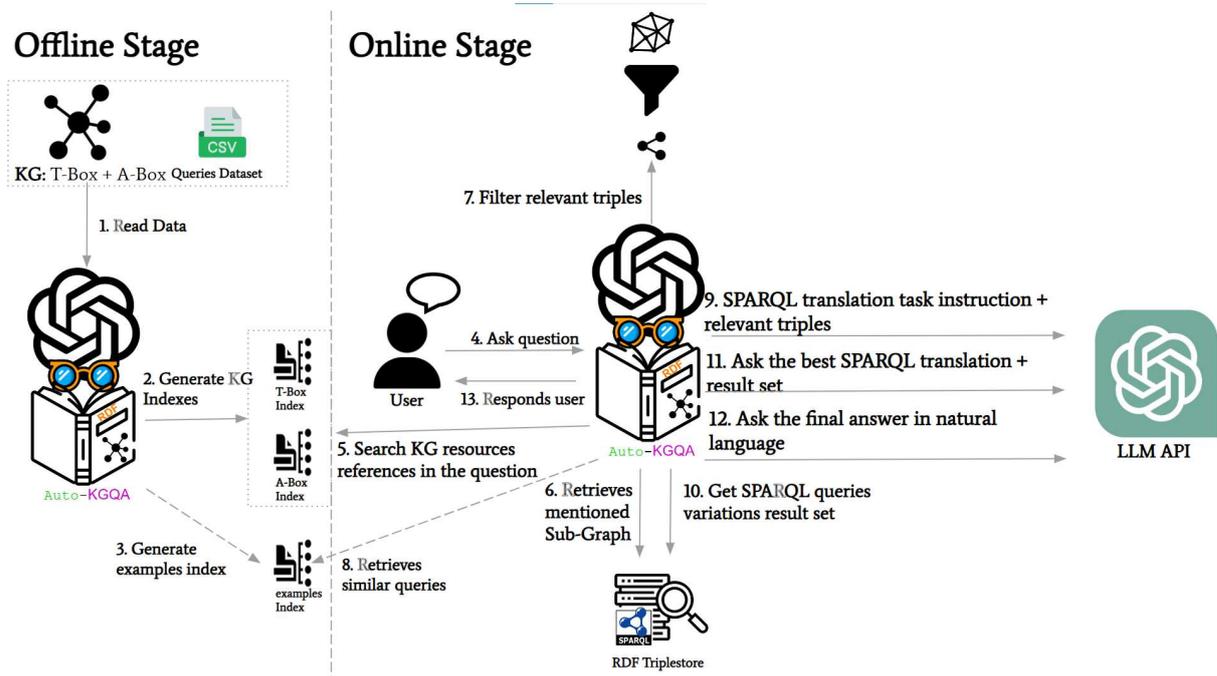
Fonte: Próprio autor.

<sup>1</sup> <https://github.com/CaioViktor/Auto-KGQA>

## 4.1 Framework Outline

O *framework* proposto recebe como entrada uma questão em linguagem natural sobre um KG e retorna como saída uma resposta em linguagem natural gerada com base no resultado da execução da consulta SPARQL traduzida da questão original sobre o KG. O Fluxo do *framework* é dividido em duas etapas (1) etapa *off-line* e (2) etapa *online* como mostrado na Figura 10.

Figura 10 – Workflow do Framework Auto-KGQA.



Fonte: Próprio autor.

Na etapa *offline* são construídos dois índices com o objetivo de ajudar a mapear termos da pergunta de entrada para termos do KG. Além disso, opcionalmente, nesta etapa pode ser construído um índice para auxiliar na tarefa de selecionar exemplos de consultas similares já conhecidas. O processo seguido na etapa *offline* apresentado na Figura 10 é como se segue:

1. O *framework* consulta os respectivos *endpoints* contendo o T-Box e o A-box (ambos somados totalizam o grafo KG). Para o T-Box é capturada a lista de *labels*, quantidade de triplas envolvidas e URI de cada classe e propriedade. Por sua vez, para o A-Box (instâncias) é capturada a lista de *labels*, quantidades de triplas envolvidas e URI para todos os recursos contidos no KG (exceto classes e propriedades). Este processo é explicado em mais detalhes na Seção 4.2.2.1. Além disso, o *framework*, opcionalmente, também pode ler o *Queries Dataset* com os exemplos de perguntas conhecidas;

2. Nesta etapa são criados dois índices que são utilizados na etapa online. O primeiro índice gera mapeamentos das *labels* de classes e propriedades para suas URIs, deste modo atuando como um mapeamento de termos da linguagem natural para recursos do T-Box. O segundo índice gera mapeamentos das *labels* de recursos da A-Box para suas URIs, deste modo atuando como um mapeamento de termos da linguagem natural para recursos do A-Box. Este processo é explicado em mais detalhes na Seção 4.2.2.2;
3. Opcionalmente, caso hajam exemplos de consultas (pares de perguntas em linguagem natural e suas respectivas consultas SPARQL) existentes no *Queries Dataset*, o *framework* constrói um índice baseado em distância de vetores destes exemplos. Este índice é utilizado para auxiliar a tarefa de encontrar exemplos de consultas conhecidas similares à pergunta sendo realizada na etapa *online*.

A fase *online* recebe uma pergunta em linguagem natural  $Q$  como entrada e retorna como saída sua resposta, também em linguagem natural,  $R$ . O processo seguido na etapa *online* apresentado na Figura 10 é como se segue:

4. Usuário realiza uma pergunta  $Q$  em linguagem natural via Interface de Chat Web ou API HTTP;
5. O *framework* realiza uma busca por combinações de palavras que estejam contidas nos índices gerados na etapa *offline*, buscando assim referências aos termos do KG. Nesta etapa são retornadas apenas as referências que alcançam um grau mínimo de confiança de acordo com a heurística apresentada na Seção 4.3.2.1;
6. Em seguida, para cada referência aos recursos do KG encontrada na etapa anterior, são recuperadas do KG as triplas em que o recurso participa. Este mesmo processo é executado também para seus vizinhos e os vizinhos destes de maneira recursiva, até  $N$  vezes, sendo  $N$  um valor configurável. Este processo gera um sub-grafo de  $KG$  chamado de  $KG'$  contendo as triplas nas proximidades dos termos referenciados na pergunta. Este processo é explicado em mais detalhes na Seção 4.3.2.2;
7. Em seguida, as triplas recuperadas inicialmente são filtradas para selecionar apenas as triplas que são de fato relevantes para a pergunta. Como heurística para medir a relevância de uma tripla, é adotada a medida da distância entre o vetor de *embeddings* da pergunta original e o vetor de *embeddings* da verbalização da tripla, onde quanto menor esta distância, maior será a relevância da tripla para a pergunta. Este processo gera um sub-grafo de  $KG'$  chamado de  $KG''$  contendo apenas as triplas relevantes para a pergunta. Este

processo é explicado em mais detalhes na Seção 4.3.2.3;

8. Como etapa opcional, o índice de exemplos de consultas pode ser consultado para recuperar consultas similares à  $Q$ . Estes exemplos atuarão numa abordagem *few-shot learning*, fornecendo pistas ao LLM sobre como realizar a tradução de  $Q$  para sua consulta SPARQL;
9. Em seguida, o LLM é solicitado para gerar  $n$  possíveis traduções  $Q_1, \dots, Q_n$  para SPARQL de  $Q$  sobre  $KG''$  como referência de contexto. Este processo é explicado em mais detalhes na Seção 4.3.3;
10. Em seguida, cada consulta  $Q_1, \dots, Q_n$  é executada sobre o *triplestore* contendo  $KG$  e são recuperadas amostras de seus resultados  $R_1, \dots, R_n$ . Nesta etapa são desconsideradas consultas mal-formadas. Este processo é explicado em mais detalhes na Seção 4.3.3;
11. Após isto, o LLM é solicitado para escolher a consulta  $Q_r$  que melhor representa  $Q$ , recebendo como entrada a pergunta  $Q$  e as consultas  $Q_1, \dots, Q_n$  acompanhadas de seus respectivos resultados  $R_1, \dots, R_n$ . Este processo é explicado em mais detalhes na Seção 4.3.3;
12. Uma vez tendo escolhido  $Q_r$ , o LLM é finalmente solicitado para gerar a resposta final  $R$  em linguagem natural para a pergunta do usuário, recebendo  $Q$ ,  $Q_r$  e  $R_r$ . Este processo é explicado em mais detalhes na Seção 4.3.4;
13. Por fim,  $R$  é retornado para o usuário formatado seguido o padrão *Markdown* (XIE *et al.*, 2018).

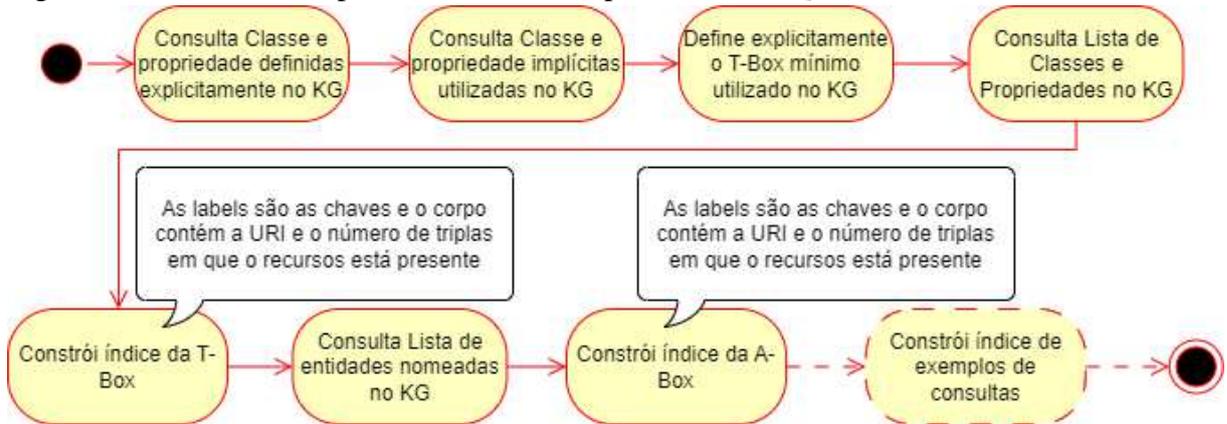
Nas sessões a seguir são dados mais detalhes sobre os processos executados durante as etapas offline e online, Sessões 4.2 e 4.3 respectivamente.

## 4.2 Etapa Offline

Durante a etapa offline são geradas estruturas de dados auxiliares que darão suporte ao processamento eficiente de consultas em tempo de execução online. Conceitualmente, o algoritmo seguido nesta etapa é composto por oito passos, como apresentados na Figura 11.

A Seção 4.2.1 apresenta a extração do T-Box mínimo do KG. Por sua vez, a Seção 4.2.2 apresenta o processo de construção de índices de recursos nomeados do KG. Por fim a Seção 4.2.3 apresenta o processo de geração do índice de exemplos.

Figura 11 – Passos da etapa offline realizados pelo Auto-KGQA



Fonte: elaborada pelo próprio autor.

#### 4.2.1 Extração do T-Box mínimo

Tendo em vista a natureza *schemaless* de KGs RDF, onde não é obrigatória a definição de classes e propriedades existentes, inicialmente o *framework* executa uma etapa de extração do T-Box mínimo que descreve tais classes e propriedades. Seja  $TBox_{min}$ , o KG RDF contendo o T-Box mínimo de KG,  $TBox_{min}$  é formado pelo conjunto de triplas que definem as classes e propriedades existentes em KG. Além disso, cada classe e propriedade é acompanhada de suas *rdfs:label*, *rdfs:comment*, *rdfs:subClassOf* (no caso de classes), *rdfs:subPropertyOf*, *rdfs:domain* e *rdfs:range* (no caso de propriedades).

Neste processo são extraídos termos explicitamente definidos no T-Box, seguindo as consultas apresentadas nos Código-fontes 1 e 3, bem como os termos implícitos, seguindo as consultas apresentadas nas listagens 2 e 4. Tal processo permite a correta listagem de todas as classes e propriedades que de fato existem em KG, inclusive aquelas reusadas, cujo em geral, não são definidas no T-Box. No caso onde um termo não possui uma *rdfs:label* explicitamente definida, um valor para esta propriedade é inferido com base no URI do termo.

Além disso, durante a extração de  $TBox_{min}$ , para cada termo é calculada sua relevância em KG. Sendo a relevância de uma classe o número de instâncias pertencentes a esta. Por sua vez, a relevância de uma propriedade é calculada como o número triplas em que esta aparece em KG.

$TBox_{min}$  é utilizado pelo *framework* em dois momentos chave (1) na criação do índice do T-Box que associa os nomes de classes e propriedades do T-Box aos seus URIs, auxiliando na identificação de referências aos termos do T-Box em consultas. (2) durante a filtragem das triplas relevantes para a questão do usuário, onde as triplas de  $TBox_{min}$  são sempre

passadas como triplas a serem consideradas, visando garantir que o algoritmo de seleção de triplas sempre tenha uma visão global do “esquema” seguido por KG. Deste modo, caso não tenha sido recuperado uma caminho conectando diretamente dois recursos citados na questão através do A-Box, existe a possibilidade de conectá-los através de  $TBox_{min}$ , tendo em vista que cada recurso estará conectado a uma classe pela propriedade  $rdf:type$ , onde por sua vez, as classes estão conectadas em si por propriedades  $rdfs:subClassOf$ ,  $rdfs:domain$  e  $rdfs:range$ .

A baixo é descrito conceitualmente o processo de extração de  $TBox_{min}$ :

1. Recupera lista de classes explícitas (*labels*, comentários, superclasses);
2. Recupera lista de classes implícitas (*labels*, comentários, superclasses);
3. Recupera número de instâncias para cada classe (relevância);
4. Recupera lista de propriedades explícitas (*labels*, comentários, super-propriedades, *domain*, *range*);
5. Recupera lista de propriedades implícitas (*labels*, comentários, super-propriedades, *domain*, *range*);
6. Recupera número de triplas em que cada propriedade aparece (relevância).

Código-fonte 1 – Consulta SPARQL para recuperar classes explícitas do T-Box

```

1 SELECT DISTINCT ?term ?label ?comments ?superclass WHERE {
2   {
3     ?term a owl:Class
4   } UNION {
5     ?term a rdfs:Class
6   }
7   OPTIONAL {
8     ?term rdfs:label ?label .
9
10  }
11  OPTIONAL {
12    ?term rdfs:comment ?comments
13  }
14  OPTIONAL {
15    ?term rdfs:subClassOf ?superclass

```

```

16         FILTER(?term != ?superclass)
17     }
18 }

```

### Código-fonte 2 – Consulta SPARQL para recuperar classes implícitas do T-Box

```

1 SELECT DISTINCT ?term ?label ?comments ?superclass WHERE{
2     ?a a ?term.
3     OPTIONAL{
4         ?term rdfs:label ?label.
5     }
6     OPTIONAL{
7         ?term rdfs:comment ?comments
8     }
9     OPTIONAL{
10        ?term rdfs:subClassOf ?superclass
11        FILTER(?term != ?superclass)
12    }
13 }

```

### Código-fonte 3 – Consulta SPARQL para recuperar propriedades explícitas do T-Box

```

1 SELECT DISTINCT ?term ?label ?comments ?superproperty ?
   domain ?range WHERE{
2     {
3         ?term a rdf:Property
4     }UNION{
5         ?term a owl:DatatypeProperty
6     }UNION{
7         ?term a owl:ObjectProperty
8     }UNION{
9         ?term rdfs:domain ?domain

```

```

10 }
11 UNION{
12     ?term rdfs:range ?range
13 }
14 OPTIONAL{
15     ?term rdfs:label ?label.
16 }
17 OPTIONAL{
18     ?term rdfs:comment ?comments
19 }
20 OPTIONAL{
21     ?term rdfs:subPropertyOf ?superproperty
22     FILTER(?term != ?superproperty)
23 }
24 OPTIONAL{
25     ?term rdfs:domain ?domain
26 }
27 OPTIONAL{
28     ?term rdfs:range ?range
29 }
30 }

```

#### Código-fonte 4 – Consulta SPARQL para recuperar propriedades implícitas do T-Box

```

1
2 SELECT DISTINCT ?term ?label ?comments ?superproperty ?
   domain ?range WHERE{
3     ?a ?term ?b.
4     FILTER(?term!=rdf:type)
5     FILTER(?term!=rdfs:subPropertyOf)
6     FILTER(?term!=rdfs:subClassOf)
7     FILTER(?term!=rdfs:domain)

```

```

8      FILTER(?term!=rdfs:range)
9      FILTER(?term!=rdfs:comment)
10     FILTER(?term!=owl:sameAs)
11     FILTER(?term!=owl:equivalentProperty)
12     FILTER(?term!=owl:equivalentClass)
13     OPTIONAL{
14         ?term rdfs:label ?label.
15     }
16     OPTIONAL{
17         ?term rdfs:comment ?comments
18     }
19     OPTIONAL{
20         ?term rdfs:subPropertyOf ?superproperty
21         FILTER(?term != ?superproperty)
22     }
23     OPTIONAL{
24         ?a a ?domain
25     }
26     OPTIONAL{
27         ?b a ?range
28     }
29 }

```

#### 4.2.2 Geração de Índices do KG

Durante a etapa de geração de índices do KG são executados dois principais algoritmos: (1) algoritmo de geração dos índices; (2) algoritmo de povoamento dos índices. Estes algoritmos são apresentados respectivamente nas seguintes seções 4.2.2.1 e 4.2.2.2.

#### 4.2.2.1 *GenerateIndexes Algorithm*

Durante a etapa *offline* é realizado o processo de criação de índices que serão utilizados durante a etapa *online* para identificar termos do KG na questão de entrada e , opcionalmente, o índice para selecionar exemplos de consultas similares à pergunta sendo realizada. Este processo é descrito pelo algoritmo apresentado em Algoritmo 1. O Algoritmo 1 realiza o processo de criação dos índices para os termos existentes nas T-Box (linhas 2-4) e A-Box (linhas 5-7). A linha 8 tenta recuperar os exemplos de consultas existentes no *Queries Dataset*, sendo estes usados, caso existam, na linha 10 para criar um índice baseado em distância de vetores dos exemplos de consultas conhecidas. O Algoritmo recebe como entrada o *endpoint* onde o KG está carregado e os tipos de cada índice a ser criado, denotado pelos parâmetros *t\_box\_index\_type* e *a\_box\_index\_type*.

Os índices gerados nesta etapa podem ser de três tipos:

1. Índice de pesquisa de texto completo (FULL\_TEXT): É um índice que ajuda a pesquisar trechos da *string* de entrada que correspondem parcialmente aos rótulos dos elementos do KG. Este tipo de índice possui baixo custo computacional e facilita a interpretação dos resultados. Pode ser usado tanto para o T-Box quanto para o A-Box. Este índice foi aplicado aos termos do A-Box nos experimentos descritos neste trabalho;
2. Índice de distância entre vetores de *embeddings* (VECTOR\_DISTANCE): É índice dos *embeddings* dos rótulos dos elementos do KG que ajuda a pesquisar trechos da *string* de entrada que possuem vetores de *embeddings* que são próximo aos vetores de *embeddings* dos rótulos de termos do KG indexados. Este tipo de índice tem a vantagem de retornar termos semanticamente próximos, embora sintaticamente diferentes, como sinônimos, etc. Pode ser usado no T-Box, A-Box e índice de exemplos. Este índice foi aplicado aos termos do T-Box e aos exemplos de consultas nos experimentos descritos neste trabalho;
3. *DBpedia Spotlight* (SPOTLIGHT): usa a API DBpedia Spotlight<sup>2</sup> para anotar referências a recursos da DBpedia em trechos do texto da questão de entrada. Este tipo de índice só pode ser utilizado ao realizar consultas na DBpedia, indexando apenas seu A-Box.

<sup>2</sup> <https://www.dbpedia-spotlight.org/api>

---

**Algoritmo 1: GenerateIndexes**

---

**Input:** endpoint, t\_box\_index\_type, a\_box\_index\_type**Output:** t\_box\_index, a\_box\_index, examples\_index**1 Function *GenerateIndexes(endpoint)*:**

```

    // Create T-Box index
2   t_box_terms ← endpoint.retrieveTBox();
3   t_box_index ← newIndex(t_box_index_type);
4   t_box_index ← PopulateIndex(t_box_index, t_box_terms, endpoint);
    // Create A-Box index
5   a_box_terms ← endpoint.retrieveABox();
6   a_box_index ← newIndex( a_box_index_type);
7   a_box_index ← PopulateIndex(a_box_index, a_box_terms, endpoint);
8   examples_queries ← retrieveExamples();
9   if examples_queries.length > 0 then
10  |   examples_index ← createExamplesIndex(examples_queries);
11  end
12  else
13  |   examples_index ← null;
14  end
15  return t_box_index, a_box_index, examples_index;
```

---

A função *endpoint.retrieveTBox()* (linha 2) recupera as classes e propriedades existentes no KG, onde cada linha da resposta contará com a URI de um termo (classe ou propriedade), sua *label* (quando esta existir) e o tipo do termo. A Listagem 5 apresenta a consulta SPARQL responsável por este processo. Em seguida o algoritmo cria um novo objeto para o índice da T-Box (linha 3) e o povoa por meio da função *PopulateIndex()* definida no Algoritmo 2. Processo semelhante é seguindo para criar o índice da A-Box, excetuando que a consulta a ser usada na função *endpoint.retrieveABox()* é aquela apresentada na Listagem 6.

#### 4.2.2.2 *PopulateIndex Algorithm*

O Algoritmo 2 recebe como entrada a lista de termos existentes no índice a ser povoado e gera como resultado este índice. Para cada termo o algoritmo decide a chave de busca que será usada (linha 9), sendo está preferencialmente a *label* do termo definida explicitamente (linha 4)

ou sua *label* inferida de sua URI (7). Cada elemento do índice conta com elementos como a URI do termo no KG, *label* original, seu tipo tipo (*class*, *property* ou *resource*) e o número de triplas em que o termo aparece no KG (*triples\_participate* obtido na linha 10).

O texto contido na chave de busca definido na linha 9 passa por um processo de normalização, onde possíveis termo em notação *camel case*<sup>3</sup> são decodificados; então, o texto é convertido inteiramente para letras minúsculas e por fim separado em *tokens*, que por sua vez serão substituídos por suas versões lematizadas<sup>4</sup>.

---

**Algoritmo 2: PopulateIndex**


---

**Input:** index, terms, endpoint

**Output:** index

```

1 Function PopulateIndex(index, terms, endpoint):
2   for each term in terms do
3     if term.label ≠ then
4       label ← term.label;
5     end
6     else
7       label ← inferLabelFromURI(term.uri);
8     end
9     indexKey ← normalize_input(label);
10    term_in_triples_count ← endpoint.countTriples(term.uri);
11    index.add(indexKey, {'uri': term.uri, 'label': term.label, 'type': term.type,
12      'triples_participates': term_in_triples_count});
13  end
14  return index;

```

---

A seguir são apresentadas as consultas SPARQL utilizadas para recuperar os termos da T-Box e A-Box que serão indexados.

Código-fonte 5 – Consulta SPARQL para recuperar termos, tipos e rótulos da T-Box

```

1 SELECT DISTINCT ?term ?type ?label ?property ?qtd WHERE {
2   {
3     ?term a owl:Class .

```

<sup>3</sup> <https://pt.wikipedia.org/wiki/CamelCase>

<sup>4</sup> <https://spacy.io/api/lemmatizer>

```
4         BIND("class" as ?type)
5     {
6         SELECT ?term (COUNT(*) as ?qtd) WHERE{
7             [] rdf:type ?term.
8         } GROUP BY ?term
9     }
10 }UNION{
11     ?term a rdfs:Class.
12     BIND("class" as ?type)
13     {
14         SELECT ?term (COUNT(*) as ?qtd) WHERE{
15             [] rdf:type ?term.
16         } GROUP BY ?term
17     }
18 }UNION{
19     ?term a rdf:Property.
20     BIND("property" as ?type)
21     {
22         SELECT ?term (COUNT(*) as ?qtd) WHERE{
23             [] ?term [].
24         } GROUP BY ?term
25     }
26 }UNION{
27     ?term a owl:DatatypeProperty.
28     BIND("property" as ?type)
29     {
30         SELECT ?term (COUNT(*) as ?qtd) WHERE{
31             [] ?term [].
32         } GROUP BY ?term
33     }
34 }
35 UNION{
```

```

36     ?term a owl:ObjectProperty.
37     BIND("property" as ?type)
38     {
39         SELECT ?term (COUNT(*) as ?qtd) WHERE{
40             [] ?term [].
41         } GROUP BY ?term
42     }
43 }
44 FILTER(!REGEX(STR(?term),"http://www.w3.org/2002/07/owl
    #","i"))
45 FILTER(!REGEX(STR(?term),"http://www.w3.org/2000/01/rdf
    -schema#","i"))
46 FILTER(!REGEX(STR(?term),"http://www.w3.org/1999/02/22-
    rdf-syntax-ns#","i"))
47 FILTER(!REGEX(STR(?term),"http://www.w3.org/2001/
    XMLSchema#","i"))
48 FILTER(!REGEX(STR(?term),"http://www.ontotext.com/","i"
    ))
49 OPTIONAL{
50     ?term rdfs:label ?label1.
51     FILTER(LANG(?label1) = "en")
52
53 }
54     BIND(COALESCE(?label1,?term) as ?label)
55 }

```

#### Código-fonte 6 – Consulta SPARQL para recuperar termos, tipos e rótulos da A-Box

```

1 SELECT DISTINCT ?term ?type ?label ?property ?qtd WHERE {
2     ?term ?p ?o.
3
4     FILTER(!REGEX(STR(?term),"http://www.w3.org/2002/07/owl

```

```
5     #", "i"))
6 FILTER(!REGEX(STR(?term), "http://www.w3.org/2000/01/rdf
7     -schema#", "i"))
8 FILTER(!REGEX(STR(?term), "http://www.w3.org/1999/02/22-
9     rdf-syntax-ns#", "i"))
10 FILTER(!REGEX(STR(?term), "http://www.w3.org/2001/
11     XMLSchema#", "i"))
12 FILTER(!REGEX(STR(?term), "http://www.ontotext.com/", "i"
13     ))
14
15 FILTER NOT EXISTS {
16     {
17         ?term a owl:Class.
18     } UNION {
19         ?term a rdfs:Class.
20     } UNION {
21         ?term a rdf:Property.
22     } UNION {
23         ?term a owl:DatatypeProperty.
24     }
25     UNION {
26         ?term a owl:ObjectProperty.
27     }
28 }
29
30 OPTIONAL {
31     ?term ?property ?label1.
32     FILTER(
33         ?property = rdfs:label ||
34         ?property = foaf:name ||
35         ?property = skos:prefLabel ||
36         ?property = dc:title ||
```

```

32         ?property = dcterms:title ||
33         ?property = dbo:name ||
34         ?property = dbp:name ||
35         ?property = dbo:name ||
36         ?property = dbp:name
37     )
38     FILTER(LANG(?label1) = "en")
39
40 }
41 BIND("resource" as ?type)
42 BIND(COALESCE(?label1,?term) as ?label)
43 {
44     SELECT ?term (COUNT(*) as ?qtd) WHERE{
45         {?a ?b ?term.}
46         UNION
47         {?term ?c ?d.}
48     } GROUP BY ?term
49 }
50 }

```

### 4.2.3 Geração de Índice de Exemplos

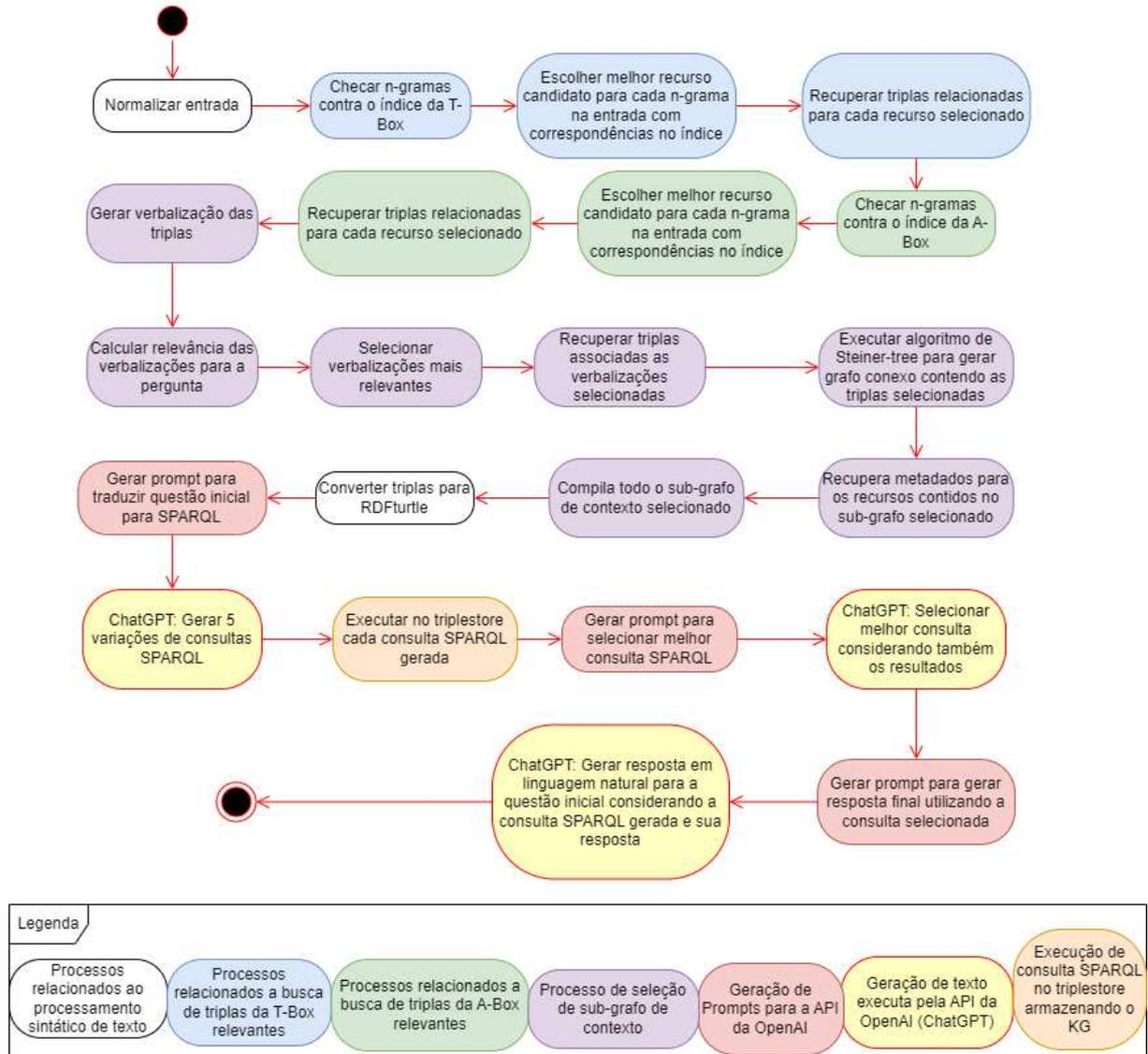
Tendo como objetivo o uso da abordagem *few-shot learning*, é construído um índice para auxiliar na tarefa de encontrar exemplos de questões conhecidas mais similares a questão sendo realizada. Este índice caracterizasse como um índice de distância entre vetores de *embeddings*, como apresentado na Seção 4.2.2.1. Seu processo de criação é semelhante ao descrito anteriormente, excetuando este índice tem como chaves as perguntas em linguagem natural e possui como corpo um fragmento de *prompt* contendo a pergunta e sua respectiva consulta SPARQL.

Deste modo, cada exemplo de consulta (pergunta-SPARQL) dá origem a um documento do índice, considerando apenas exemplos avaliados como corretos.

### 4.3 Etapa Online

Na etapa online são respondidas as perguntas feitas pelo usuário. A Figura 12 apresenta os passos conceituais realizados nesta etapa.

Figura 12 – Passos da etapa online realizados pelo Auto-KGQA



Fonte: elaborada pelo próprio autor.

Nesta seção a consulta **“What is John’s age?”** será utilizada para exemplificar o funcionamento do algoritmo seguido por Auto-KGQA para responder questões em linguagem natural.

### 4.3.1 Algoritmo para Responder Questões em Linguagem Natural

O algoritmo 3 recebe como entrada a questão em NL, o *endpoint* onde o KG está carregado, os índices gerados na fase *offline*, o número máximo de saltos por propriedade, e o número máximo de arestas por propriedade.

O algoritmo 3 é dividido em três sub-algoritmos, responsáveis respectivamente por:

1. extrair do KG apenas as triplas que serão relevantes para escrever a consulta SPARQL, considerando tanto T-Box quanto A-Box (Linha 1) apresentado na Seção 4.3.2;
2. gerar a consulta SPARQL que melhor traduza a pergunta do usuário, considerando tanto a consulta quanto o resultado de sua execução no KG (Linha 2) apresentado na Seção 4.3.3;
3. gerar a resposta final em LN (Linha 3) apresentado na Seção 4.3.4.

---

#### Algoritmo 3: AnswerQuestion

---

**Input:** question, endpoint, index\_t\_box, index\_a\_box, edge\_hops,  
max\_edges\_by\_property

**Output:** answer

- 1 relevant\_sub\_graph  $\leftarrow$  ExtractRelevantSubgraph(question, endpoint, index\_t\_box,  
index\_a\_box, edge\_hops, max\_edges\_by\_property);
  - 2 sparql\_query  $\leftarrow$  TranslateQuestion(question, relevant\_sub\_graph, endpoint);
  - 3 answer  $\leftarrow$  GenerateFinalAnswer(question, sparql\_query, endpoint);
  - 4 **return** answer;
- 

### 4.3.2 Algoritmo de Extração do Sub-grafo Relevante para a Questão

O algoritmo 4 primeiro normaliza a entrada (Linha 2), usando o mesmo processo aplicado às chaves de pesquisa no estágio *offline*. O resultado é uma sequência de *tokens* lematizados que representa a entrada original.

---

**Algoritmo 4:** ExtractRelevantSubgraph
 

---

**Input:** question, endpoint, index\_t\_box, index\_a\_box, edge\_hops,  
max\_edges\_by\_property

**Output:** relevant\_sub\_graph

```

1 visited_nodes ← set();
2 normalized_question ← normalize_input(question);
3 t_box_resources ← RetrieveResources(normalized_question, index_t_box);
4 t_box_triples ← RetrieveTriples(endpoint, visited_nodes, t_box_resources,
    visited_nodes, edge_hops, max_edges_by_property) + getTBoxMin();
5 a_box_resources ← RetrieveResources(normalized_question, index_a_box);
6 a_box_triples ← RetrieveTriples(endpoint, visited_nodes, a_box_resources,
    visited_nodes, edge_hops, max_edges_by_property);
7 full_triples ← t_box_triples + a_box_triples;
8 explicit_resources ← t_box_resources + a_box_resources;
9 triples ← FilterOutTriples(full_triples, question, explicit_resources);
10 relevant_sub_graph ← convert_to_turtle(triples);
11 return relevant_sub_graph;
```

---

Em seguida, ele recupera os possíveis termos do KG referenciados na pergunta em LN de entrada (Linhas 3 e 5), descrito em detalhes nos algoritmos 5 e 6. No exemplo discutido, são encontradas as seguintes referências explícitas aos elementos do KG:

```

[‘http://www.example.lirb.com/JohnDoe’,
 ‘http://www.example.lirb.com/hasAge’]
```

Então o algoritmo usa esses termos para selecionar fragmentos do KG (Linhas 4 e 6) do T-Box e do A-Box, selecionando todas as triplas dos recursos referenciados por estes termos e dos recursos em sua vizinhança com profundidade de até  $dge\_hops$ . Onde no caso das triplas do T-Box (linha 4), também são recuperadas todas as triplas do grafo mínimo do T-Box,  $TBox_{min}$ , como descrito na Seção 4.2.1. No exemplo estudado,  $dge\_hops$  é igual a 2, capturando assim os vizinhos diretos e seus vizinhos. Em seguida os fragmentos são concatenados (Linha 7), formando um único sub-grafo RDF. A linha 8 gera a lista completa de recursos mencionados explicitamente na pergunta.

Abaixo está o sub-grafo recuperado usando os dois nós como ponto de partida, porém, devido ao seu tamanho, o sub-grafo foi cortado até o limite de 1228 caracteres (tamanho escolhido por ser o tamanho exato do sub-grafo resultante após filtragem), enquanto o sub-grafo completo é formado por 9243.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ns1: <http://www.arida.ufc.br/ontology/timeline/> .
@prefix ns2: <http://www.example.lirb.com/> .
@prefix ns3: <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix ns4: <https://raw.githubusercontent.com/CaioViktor/
LirB/main/lirb_ontology.ttl/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ns2:Animal a rdfs:Class,
    owl:Class ;
    rdfs:label "Animal" ;
    rdfs:comment "A living organism that is not a human
being." ;
    rdfs:subClassOf ns2:LivingBeing .

ns2:Dog a rdfs:Class,
    owl:Class ;
    rdfs:label "Dog" ;
    rdfs:comment "A domesticated carnivorous mammal that is
kept as a pet or used for various working purposes." ;
    rdfs:subClassOf ns2:Animal,
        ns2:LivingBeing .

ns2:Female a rdfs:Class,
    owl:Class ;
```

```

rdfs:label "Female" ;
rdfs:comment "A female human being." ;
rdfs:subClassOf ns2:LivingBeing,
    ns2:Person .

ns2:LivingBeing a rdfs:Class,
    owl:Class ;
rdfs:label "Living Being" ;
rdfs:comment "An organism that is alive ..."

```

Em seguida, o Algoritmo 7 é chamado para filtrar apenas as triplas que são relevantes para a questão (Linha 9). Finalmente, o algoritmo converte as triplas relevantes para serialização Turtle RDF (Linha 10), gerando um único sub-grafo passado para o LLM para ser usado na geração de consultas SPARQL. Abaixo segue o sub-grafo filtrado apenas com as triplas consideradas mais relevantes para a questão, totalizando 1228.

```

@prefix ns1: <http://www.example.lirb.com/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ns1:John a ns1:LivingBeing,
    ns1:Person ;
rdfs:label "John Doe",
    "Jonny" ;
ns1:hasAge 25 ;
ns1:hasBirthday "1998-01-01"^^xsd:date ;
owl:sameAs ns1:JohnDoe .

ns1:hasAge a rdf:Property,
    owl:DatatypeProperty ;
rdfs:label "has age" ;

```

```

    rdfs:domain ns1:LivingBeing ;
    rdfs:range rdfs:Literal,
        xsd:integer .

ns1:hasBirthday a rdf:Property,
    owl:DatatypeProperty ;
    rdfs:label "has birthday" ;
    rdfs:domain ns1:LivingBeing ;
    rdfs:range xsd:date .

ns1:JohnDoe a ns1:LivingBeing,
    ns1:Person ;
    rdfs:label "John Doe",
        "Jonny" ;
    ns1:hasAge 25 ;
    ns1:hasBirthday "1998-01-01"^^xsd:date .

ns1:Person a owl:Class ;
    rdfs:label "Person" ;
    rdfs:comment "A human being. " .

ns1:LivingBeing a owl:Class ;
    rdfs:label "Living Being" ;
    rdfs:comment "An organism that is alive and capable of
performing vital functions such as growth, metabolism,
and reproduction. " .

```

Ao comparar os dois sub-grafos é nítido que o segundo conta com as triplas necessárias para responder a questão, tomando um menor número de caracteres.

#### 4.3.2.1 Algoritmo para Identificar Citações a Recursos do KG na Questão

Algoritmo 5 identifica referências a elementos do KG pesquisando a sequência de tokens normalizados da pergunta de entrada para encontrar possíveis referências aos termos do KG nó índice sendo passado como entrada.

Primeiro, a função *GetPossibleMatches()* (Linha 1) separa os tokens em n-gramas, e procura-os nos índices T-Box ou A-Box criado na etapa offline. A criação de n-gramas candidatos é realizada através de um processo de correspondência de padrões utilizando uma janela deslizante de tamanho decrescente (FIGUEIREDO, 2007). Neste processo, uma janela, inicialmente com o tamanho  $n$  igual ao número de *tokens* na sequência inteira, é criada da concatenação dos  $n$  tokens consecutivos, iniciando-se no *token* mais a esquerda da sequência, onde em cada iteração a janela de tamanho  $n$  será deslocada em um *token* para a direita até atingir o final da sequência, onde o processo será reiniciado (novamente a partir da esquerda) considerando novas janelas de tamanho  $n-1$ , até parar, quando a janela for de tamanho 0.

Para cada n-grama encontrado em um índice, O algoritmo 5 escolhe apenas um termo candidato no KG como segue. Primeiro calcula as pontuações de relevância dos termos candidatos (Linha 6). Essa pontuação considera o valor de similaridade (peso 2) entre as strings da chave de pesquisa e o n-grama em questão, bem como a quantidade de triplas (peso 1) em que o recurso aparece no KG. Esses valores já estão normalizados para uma escala entre 0 e 1, e sua média ponderada é calculada, resultando em uma relevância do termo entre 0 e 1. Algoritmo 5 seleciona o termo candidato com maior relevância (Linha 8), e retorna sua URI (Linha 10).

Por último, o Algoritmo 5 retorna a lista de URIs dos termos relevantes para a pergunta de entrada (Linha 12).

---

**Algoritmo 5: RetrieveResources**


---

**Input:** normalized\_question, index

**Output:** selected\_resources

```

1 matches ← GetPossibleMatches(normalized_question, index);
2 selected_resources ← [];
3 for match in matches do
4   | scores ← [];
5   | for hit in match.hits_index do
6   |   | scores[hit] ← (2 * hit.similarity_to_input + hit.triples_participates) / 3;
7   | end
8   | selected_hit ← argmax(scores);
9   | resource_uri ← selected_hit.uri;
10  | selected_resources.add(resource_uri);
11 end
12 return selected_resources;

```

---

#### 4.3.2.2 Algoritmo para Recuperar Potenciais Triplas Relevantes no endpoint

O Algoritmo 6 recupera as triplas que definem o fragmento do KG. Para cada termo relevante  $t$  que o Algoritmo 5 retorna, ele recupera do KG todas as triplas em que o URI de  $t$  aparece como sujeito ou objeto (Linha 5). Portanto, captura todas as propriedades de e para um nó. Porém, as triplas que não agregam valor semântico relevante ao recurso são filtradas. Alguns exemplos de triplos excluídos são aqueles que definem o termo como *rdfs:Resource* ou *owl:Thing*, triplos que indicam auto-referências resultante da aplicação de axiomas, como triplos que indicam que uma classe é uma subclasse (*rdfs:subClassOf*) dela mesma, uma classe é equivalente (*owl:equivalentClass*) a si mesma, ou um recurso é igual (*owl:sameAs*) a si mesmo.

Além das triplas diretamente relacionados a um termo  $t$ , O algoritmo 6 também captura as triplas relacionados aos vizinhos de  $t$  e os vizinhos dos vizinhos de  $t$ , em um processo recursivo (Linha 8), até uma profundidade definida pelo parâmetro “edge\_hops”.

---

**Algoritmo 6: RetrieveTriples**


---

**Input:** endpoint, visited\_nodes, selected\_resources, n\_hops

**Output:** triples

```

1 triples ← string();
2 for selected_resource in selected_resources do
3   if not visited_nodes.contains(selected_resource) then
4     visited_nodes.add(selected_resource);
5     triples ← triples + endpoint.get_triples(selected_resource, edge_hops,
6       visited_nodes);
6   end
7 end
8 triples ← RetrieveTriples(endpoint, visited_nodes, selected_resources,
9   visited_nodes, n_hops = n_hops - 1);
9 return triples;

```

---

#### 4.3.2.3 Algoritmo de Filtragem de Triplas Relevantes

O algoritmo 7 recebe como entrada um conjunto de triplas recuperadas do KG, na vizinhança dos nós correspondendo aos termos referenciados na consulta do usuário. Ele seleciona apenas as triplas que são relevantes para a consulta, com base na intuição de que os *embeddings* de triplas mais relevante para a questão estariam mais próximos dos *embeddings* da questão.

---

**Algoritmo 7:** FilterOutTriples Algorithm
 

---

**Input:** triples, question, explicit\_resources

**Output:** filtered\_triples

```

1 selected_triples ← getResourcesDefinitionTriples(triples,explicit_resources);
2 verbalizations ← CreateVerbalizations(triples);
3 verbalizations_index ← newIndex("VECTOR_DISTANCE");
4 for verbalization in verbalizations do
5   | verbalizations_index.add(verbalization.text, { 'triples': verbalization.triples });
6 end
7 relevant_verbalizations ← GetRelevantVerbalizations(question, verbalizations_index);
8 relevant_triples ← [];
9 for verbalization in relevant_verbalizations do
10  | relevant_triple ← getRelevantTriple(verbalization.triples, question);
11  | relevant_triples.add(relevant_triple);
12 end
13 selected_triples ← getConnectedGraphTriples(triples, selected_triples);
14 return selected_triples;
```

---

Embora seja possível usar técnicas, como Graph Neural Networks (GNN) (WU *et al.*, 2020), para mapear recursos e triplas RDF para *embeddings*, o espaço vetorial dos *embeddings* pode ser diferente do espaço vetorial dos *embeddings* da questão em NL. O algoritmo 7 usa em vez disso uma “pseudo-verbalização” para cada conjunto de triplos que compartilham o mesmo sujeito e propriedade, variando apenas o objeto. Este agrupamento visa evitar que os vetores mais próximos da questão sejam apenas valores diferentes de uma propriedade com vários valores, aumentando assim a variedade das triplas selecionadas. Portanto, para cada grupo de triplas, é gerada uma única frase que representa uma “pseudo-verbalização” do grupo.

As verbalizações são sentenças em LN com a estrutura “*Sujeito Verbo Complemento*”. O sujeito e o verbo da frase são formados por os rótulos dos recursos nas triplas originais do grupo. O complemento é uma lista de valores que representam os diferentes objetos de cada tripla do grupo, separados por “vírgulas”. Quando o objeto é um recurso RDF, é representado na frase pelo seu rótulo; quando é literal, é usado diretamente o seu valor na frase. A seguir são apresentados conjunto de triplas e suas verbalizações resultantes

["<http://www.example.lirb.com/JohnDoe>

```

<http://www.example.lirb.com/hasAge>
25."] -> 'John Doe has age 25'

["<http://www.example.lirb.com/JohnDoe>
<http://www.example.lirb.com/knows>
<http://www.example.lirb.com/Jane>.",
"<http://www.example.lirb.com/JohnDoe>
<http://www.example.lirb.com/knows>
<http://www.example.lirb.com/Tom>."] -> 'John Doe knows Jane, Tom.'

```

Após a seleção das triplas mais relevantes para a questão, um algoritmo de *steiner-tree* (HWANG; RICHARDS, 1992) é usado para gerar um grafo conectado contendo essas triplas sobre os sub-grafo do KG recuperado. Algoritmos 7 e 8 detalham a filtragem das triplas e o processo de verbalização das triplas.

O algoritmo 7 recebe como entrada um conjunto de triplas recuperadas do KG mencionadas na pergunta do usuário e as triplas relacionadas a estes termos. Inicialmente recupera (Linha 1) as triplas que definem cada um dos recursos (rdf:type e rdfs:label) mencionados explicitamente na questão obtida na etapa de seleção de recursos. Isso se reflete nas seguintes triplas no exemplo estudado:

```

@prefix ns1: <http://www.example.lirb.com/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ns1:JohnDoe a ns1:LivingBeing,
            ns1:Person ;
            rdfs:label "John Doe",
            "Jonny" .

ns1:hasAge a rdf:Property,

```

```

    owl:DatatypeProperty ;
    rdfs:label "has age" ;
    rdfs:domain ns1:LivingBeing ;
    rdfs:range rdfs:Literal,
        xsd:integer .

```

Em seguida, o Algoritmo 8 (Linha 2) é chamado para criar as verbalizações da triplas. Dando prosseguimento, ele cria um índice para calcular a distância dos *embeddings* das verbalizações para a pergunta do usuário (Linha 3) e o preenche com as verbalizações geradas (Linhas 4-6).

Em seguida, o algoritmo recupera as verbalizações relevantes à pergunta (Linha 7); este processo é realizado pela função *GetRelevantVerbalizations()*, que recebe como entrada a pergunta e o índice de verbalizações e calcula as verbalizações de acordo com quais vetores de *embeddings* estão mais próximos do vetor de pergunta. A função *GetRelevantVerbalizations()* também recebe um limite para a relevância mínima dos resultados (*min\_relevance\_threshold*), além do número máximo de resultados retornados (*maximun\_number\_results*), que são hiperparâmetros do algoritmo e foram omitidos por questão de simplicidade. A seguir, o Algoritmo 7 recupera, para cada verbalização, as triplas mais relevantes pertencentes ao grupo das triplas que originou a verbalização (Linhas 9-12). Este processo é feito pela função *getRelevantTriple()* (Linha 10), que recebe como entrada as triplas do grupo e a pergunta, retornando apenas a tripla cujo vetor de *embeddings* do objeto da tripla está mais próximo da questão. As verbalizações recuperadas nesta fase acompanhadas de suas triplas para o exemplo do estudo estão abaixo:

```

'John Doe has age 25':
    <http://www.example.lirb.com/JohnDoe>
    <http://www.example.lirb.com/hasAge>
    25.

'John has birthday 1998-01-01':
    <http://www.example.lirb.com/John>
    <http://www.example.lirb.com/hasBirthday>
    "1998-01-01"^^http://www.w3.org/2001/XMLSchema#date.

```

Neste exemplo em específico é possível notar que o algoritmo selecionou duas verbalizações que são relevantes para a consulta, embora a primeira já seria suficiente. Além disso, é possível notar que as triplas selecionadas são sobre recursos diferentes, deste modo precisando serem conectados.

Finalmente, o algoritmo 7 executa a função *getConnectedGraphTriples()* (Linha 13), que recebe como entrada as triplas extraídas do KG e os triplas mais relevantes para a questão, retornando um grafo conectado que passa por essas triplas relevantes. A função *getConnectedGraphTriples()* implementa um algoritmo de *steiner-tree* sobre uma visão não direcionada do grafo, passada como entrada, retornando uma *steiner-tree* que conecta as triplas relevantes passadas na entrada.

A execução do algoritmo de *steiner-tree* nos nós mencionados explicitamente no exemplo, juntamente com os nós presentes nas triplas relevantes selecionadas, revela a necessidade de incluir a propriedade *owl:sameAs* para conectar os recursos *ns1:JohnDoe* e *ns1:John*, além da inclusão da propriedade *ns1:hasBirthday* e das classes *ns1:Person* e *ns1:LivingBeing* (estas estão nas triplas de definição de recursos), resultando na inclusão das seguintes triplas:

```
@prefix ns1: <http://www.example.lirb.com/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ns1:John a ns1:LivingBeing,
          ns1:Person ;
rdfs:label "John Doe",
          "Jonny" ;
ns1:hasBirthday "1998-01-01"^^xsd:date ;
owl:sameAs ns1:JohnDoe .

ns1:hasBirthday a rdf:Property,
                 owl:DatatypeProperty ;
rdfs:label "has birthday" ;
rdfs:domain ns1:LivingBeing ;
```

```

    rdfs:range xsd:date .

ns1:JohnDoe a ns1:LivingBeing,
            ns1:Person ;
    rdfs:label "John Doe",
            "Jonny" ;
    ns1:hasAge 25 .

ns1:Person a owl:Class ;
    rdfs:label "Person" ;
    rdfs:comment "A human being. " .

ns1:LivingBeing a owl:Class ;
    rdfs:label "Living Being" ;
    rdfs:comment "An organism that is alive and capable of
performing vital functions such as growth, metabolism,
and reproduction. " .

```

O Algoritmo 8 recebe como entrada um conjunto de triplas, retornando uma lista de verbalizações para estas agrupadas por sujeito e propriedade.

Inicialmente é definida uma estrutura que mapeia os identificadores dos grupos de triplas às suas verbalizações (Linha 1). Então, para cada tripla na lista de entrada (Linha 2), o algoritmo define a qual grupo essa tripla deve pertencer (Linha 3). Cada grupo é identificado pela concatenação dos URIs do sujeito e da propriedade de suas triplas. Em seguida, é avaliado se a atual tripla faz parte de um novo grupo que ainda não foi visto (Linha 4); neste caso, cria um novo grupo na estrutura do grupo (Linha 6). Durante a criação de um novo grupo, a base da verbalização é criada, que é a concatenação dos rótulos do sujeito, propriedade e objeto da tripla atual (Linha 6); se o objeto da tripla for um literal, o atributo *label* terá o mesmo valor do literal. Em seguida, a verbalização do grupo é atualizada (Linha 7) e a tripla atual é adicionada ao grupo (Linha 8). Se o tripla atual fizer parte de um grupo existente, o texto da verbalização do grupo é atualizado, adicionando uma “vírgula” seguida do valor do rótulo do objeto no final (Linha 11). A tripla atual é adicionado à lista de triplas do grupo (Linha 12). Finalmente, a estrutura de

verbalizações é convertida para o formato de lista e é retornada (Linha 15).

---

**Algoritmo 8:** CreateVerbalizations Algorithm

---

**Input:** triples

**Output:** verbalizations

```

1 verbalizations ← HashMap();
2 for triple in triples do
3     key ← triple.subject + triple.property;
4     if not verbalizations.contains(key) then
5         verbalizations.add(key, { 'text':"", 'triples':[] });
6         triple_verbalization ← triple.subject.label + " " + triple.property.label + " " +
           triple.object.label;
7         verbalizations.get(key).text = triple_verbalization;
8         verbalizations.get(key).triples.add(triple);
9     end
10    else
11        verbalizations.get(key).text += ", " + triple.object.label;
12        verbalizations.get(key).triples.add(triple);
13    end
14 end
15 return verbalizations.toList();

```

---

### 4.3.3 Algoritmo de Text-to-SPARQL

O Algoritmo 9 é responsável por gerar a tradução SPARQL da consulta em linguagem natural. Inicialmente, opcionalmente, este algoritmo busca os exemplos de consultas similares à pergunta atual (linha 1). Em seguida, o algoritmo gera o *prompt* necessário para a tarefa de tradução de perguntas para uma consulta SPARQL sobre as triplas recuperadas do KG (Linha 2). Este *prompt* é então passado para a API do LLM, pedindo-lhe para gerar cinco variações de resposta (Linha 3). As linhas 6 a 11 extraem a consulta SPARQL de cada resposta LLM (Linha 7), executa-a no *triplestore* aplicando um limite de 10 amostras (Linha 8). Para o exemplo estudado, todas as cinco variações geradas são iguais, mostrando que o LLM não teve ambiguidade na interpretação da questão. Abaixo estão dois exemplos das consultas geradas, embora o restante seja semelhante com pouca variação na escrita:

```

SELECT ?john ?age WHERE {
    ?john ns1:hasAge ?age .
    ?john rdfs:label ?name.
    FILTER(REGEX(?name, "John", "i"))
}
result:
[{'?John': 'http://www.example.lirb.com/JohnDoe',
  '?age': '25'},
{'?John': 'http://www.example.lirb.com/John',
  '?age': '25'}]

SELECT ?John ?age WHERE {
    ?John rdfs:label ?name.
    ?John ns1:hasAge ?age.
    FILTER(REGEX(?name, "John", "i"))
}
[{'?John': 'http://www.example.lirb.com/JohnDoe',
  '?age': '25'},
{'?John': 'http://www.example.lirb.com/John',
  '?age': '25'}]

...

```

Em seguida, é gerado o *prompt* para indicar a tarefa para o LLM escolher a melhor tradução (linha 12), sendo a linha 13 responsável por extrair a resposta do LLM para esta tarefa. Finalmente, a melhor consulta SPARQL é retornada (Linha 15) depois de ser recuperado do vetor de consulta (Linha 14).

---

**Algoritmo 9:** TranslateQuestion Algorithm
 

---

**Input:** question, triples, endpoint

**Output:** best\_sparql

```

1 queries_examples ← getQueriesExamples(question);
2 prompt_translation ← generate_prompt_translation(question, triples, queries_examples);
3 answers ← chatGPT(prompt_translation, n=5);
4 queries ← [];
5 results ← [];
6 for answer in answers do
7   sparql ← extractSPARQL(answer);
8   result ← endpoint.run_sparql(sparql, limit=10);
9   queries.add(sparql);
10  results.add(result);
11 end
12 prompt_best_query ← generate_prompt_best_query(question, queries, results);
13 best_query_index ← extract_option_best_query(chatGPT(prompt_best_query));
14 best_sparql ← queries[best_query_index];
15 return best_sparql;

```

---

#### 4.3.4 Algoritmo de Geração da Resposta Final em Linguagem Natural

Como um etapa final e opcional, o *framework* gera uma resposta final em LN para o usuário considerando a pergunta original, a consulta SPARQL gerada junto de seu conjunto de resultados.

---

**Algoritmo 10:** GenerateFinalAnswer Algorithm
 

---

**Input:** question, sparql, endpoint

**Output:** answer

```

1 result ← endpoint.run_sparql(sparql);
2 prompt_answer ← generate_prompt_final_answer(question, sparql, result);
3 answer ← chatGPT(prompt_answer, temperature=1.2);
4 return answer;

```

---

O Algoritmo 10 executa a melhor consulta SPARQL selecionada, recuperando seu conjunto completo de respostas (Linha 1). Em seguida, o *prompt* para gerar a resposta final em

LN para a pergunta do usuário é criado (Linha 2). Ao final do processo, o algoritmo usa o LLM para gerar a resposta em LN final (Linha 3), usando como base a pergunta original em NL, a consulta SPARQL selecionada, e seu conjunto de respostas.

A resposta final gerada para o exemplo estudado é dada abaixo:

```
John, specifically known as John Doe, is 25 years old.
The information is referred from data represented in
the system by URI "http://www.example.lirb.com/JohnDoe".
```

#### 4.3.5 *Templates de Prompts*

O *template de prompt* (Linha 9 do Algoritmo 3) que solicita a geração de consultas SPARQL que são variações de tradução da pergunta original em LN é o seguinte:

```
1 System:
2   Consider the following RDF graph written in Turtle
   syntax:
3     {KG_Subgraph}
4   Write a SPARQL query for the question given by the user
   following the restrictions:
5     -Use only classes and properties defined in the RDF
       graph, for this is important to use the same
       URIs for the properties and classes as defined
       in the original graph;
6     -Include all the prefixes used in the SPARQL query;
7     -Declare non-essential properties to the question
       as OPTIONAL if needed;
8     -DO NOT use specific resources in the query;
9     Declare filters on strings (like labels and names)
       as filter operations over the REGEX function
       using the case-insensitive flag.
10  Here are some sample related questions accompanied by
       their queries to help you:
11  {queries_examples}
```

```

12 User:
13   {Question}

```

O *template de prompt* começa com o sub-grafo do KG obtido a partir dos fragmentos de termos relevantes para a questão em NL. Em seguida, contém uma lista de restrições a serem seguidas para gerar consultas corretas de acordo com o KG e que são genéricos em relação aos exemplos dados. O *template de prompt* termina com a pergunta em NL.

O *template de prompt* (Linha 19 do Algoritmo 3) que solicita a seleção da variação da consulta SPARQL que melhor representa a questão em LN original é a seguinte:

```

1 System:
2   Consider the following RDF graph written in Turtle
3     syntax:
4     {KG_Subgraph}
5   Given the question: {question}
6   Select the number of the option that better represents
7     a SPARQL query for the given question:
8   ```json
9     0:{
10      "query":"```sparql
11        {SPARQL 0}
12      """,
13      "result": {Result Dataset 0}
14    },
15    ...'''
16   Use the following criteria to evaluate the options:
17   -{Restrictions List}
18   Return only the number of the selected option!

```

O *template de prompt* começa novamente com o sub-grafo KG obtido dos fragmentos de termos relevantes para a questão em NL. Ele continua com a pergunta original de NL, junto com a lista de variações de consulta SPARQL e o conjunto de resultados de suas execuções no KG. O *template de prompt* termina com uma lista de restrições como antes.

Por último, o *template* de *prompt* (Linha 23 do Algoritmo 3) para solicitar ao LLM que gere a resposta final em LN para o usuário é o seguinte:

```

1 System:
2     Use the SPARQL query and its result set as a JSON
3     object to write an answer to the user question. DO
4     NOT explain neither cite the SPARQL query and JSON
5     in yours response.
6 User question: {question};
7 SPARQL query:
8     ```sparql
9         {SPARQL choosed}
10    ```;
11 JSON result set:
12     ```json
13         {Result set}
14     ```;

```

Este prompt passa a pergunta em LN original do usuário, junto com a consulta SPARQL selecionada e o conjunto de resultados desde a sua execução no KG.

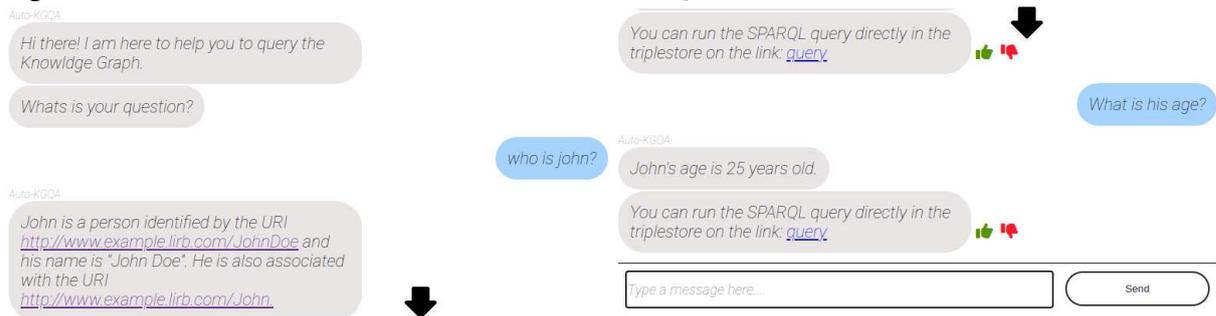
#### 4.4 Interface Web de Chat

A figura 13 apresenta uma demonstração de interação via interface Web do Auto-KGQA (a imagem originalmente uma só na vertical). Nesta demonstração é utilizado o KG *daily-life* descrito na Seção 6.1 e disponível no repositório do *github*<sup>5</sup> do projeto. No link<sup>6</sup> encontra-se um vídeo com uma demonstração em maiores detalhes.

<sup>5</sup> <https://github.com/CaioViktor/Auto-KGQA>

<sup>6</sup> <https://youtu.be/GFSHySDitzU>

Figura 13 – Interface Web de Chat com o Auto-KGQA.



Fonte: Próprio autor.

Na demonstração, inicialmente o usuário realiza a seguinte pergunta:

“Who is John?”

Após o processamento da pergunta, o sistema apresenta sua resposta, onde links para URIs de recursos apresentados na resposta são direcionados para a navegação no KG utilizando a interface LiRB partindo deste recurso. Além disso, o sistema também apresenta um link para acessar a consulta executada sobre o *endpoint* do *knowledge Graph*, permitindo ao usuário ter uma resposta mais completa ou editar a consulta original.

Como outra funcionalidade, a interface permite o *feedback* do usuário, apontando se a resposta para a pergunta foi correta ou não. Todas as perguntas processadas são salvas em um arquivo CSV, onde são também armazenadas informações intermediárias de processamento, tais como sub-grafo utilizado, variações de SPARQL geradas, além do *feedback* do usuário. Tais informações podem ser utilizadas futuramente para melhorar o desempenho do sistema.

Além disso, o sistema permite conversações multi-turno, permitindo ao usuário continuar a perguntar informações complementares à consulta anterior, ou refiná-la. Por exemplo, o usuário pode, em seguida perguntar a idade de John da seguinte maneira:

“What is his age?”

Com o sistema identificando que trata-se da idade de John anteriormente citado e trazendo sua resposta.

## 5 LIGHT RDF BROWSER

*RDF Browser* ou *Semantic Web Browser* são aplicações que localizam automaticamente metadados e montam interfaces de apontar e clicar a partir de especificações ontológicas descritas em RDF e recuperadas dinamicamente da Web Semântica (QUAN; KARGER, 2004). Deste modo, tais ferramentas podem ser reutilizadas no contexto de KGs armazenados no formato RDF.

Atualmente existem dois tipos de RDF Browsers, os *text-based* (**baseados em textos**), tais como o *OpenLink Faceted Browser*<sup>1</sup>; bem como os *visual-based* (**baseados em visual**), tais como o *LodLive*<sup>2</sup>. Os *Browsers* do primeiro tipo tendem a ser mais simples e leves, apresentando os dados como uma simples página HTML contendo listas de valores para propriedades do recurso RDF sendo explorado. Por sua vez, os do segundo tipo tendem a ser mais complexos e com mais recursos, apresentando o grafo em uma representação visual composta por nós e arestas. As interfaces baseadas em visual são o tipo que mais vem sendo explorado e recebendo novos recursos recentemente.

Embora interfaces baseadas em técnicas de visualização de grafos, tais como as apresentadas em (BIKAKIS; SELLIS, 2016), tenham um forte apelo, o uso destas na prática pode vir a ser um desafio quando navegando sobre grafos com um grande número de nós, gerando uma visualização poluída e de difícil compreensão. Além disso, interfaces deste tipo também tendem a apresentarem consultas mais complexas, o que pode vir a impactar negativamente no tempo de renderização de KGs massivos e/ou virtuais.

KGs virtuais utilizam a estratégia *ontology-based data access* (OBDA) para gerar uma visão RDF virtual sobre Bancos de Dados Relacionais (RDBs) tradicionais, realizando consultas SPARQL sobre a ontologia de domínio e traduzindo-a para consultas SQL sobre os RDBs originais, sem haver a necessidade de materializar todas as triplas RDF do KG (RODRIGUEZ-MURO *et al.*, 2013). Nesta abordagem, em geral, relações entre dois recursos no KG são traduzidas para operações de junções SQL, deste modo gerando um elevado número de junções para consultas SPARQL mais complexas, o que ocasiona um elevado custo para o processamento da consulta SQL resultante.

Deste modo, surge a necessidade do desenvolvimento de *RDF Browsers* modernos que apresentem os dados de maneira compreensível e que utilizem consultas leves, compatíveis

<sup>1</sup> <https://github.com/OpenLinkSoftware/fct>

<sup>2</sup> <http://en.lodlive.it/>

com a possibilidade de KGs virtuais. Portanto, neste artigo apresentamos *Light RDF Browser* (LiRB), uma interface leve para a navegação interativa sobre KGs RDF. LiRB apresenta uma interface Web baseada em texto, proporcionando uma experiência de navegação semelhante às páginas Web tradicionais, promovendo uma menor curva de aprendizado aos usuários. Como diferenciais, LiRB utiliza consultas simples e rápidas, sendo ideal para a navegação de KGs virtuais. Além disso, a ferramenta também permite a abstração de relacionamentos *N-ários* para representar a semântica de propriedades de relacionamentos. Ademais, a ferramenta também permite que o desenvolvedor salve consultas pré-definidas a serem apresentadas ao usuário. Por fim, LiRB também dá suporte ao recurso de exibição de *timelines* de eventos de recursos do KG.

### 5.1 LiRB: Light RDF Browser

LiRB é uma interface de navegação baseada em texto sobre KGs RDF que busca aliar recursos dos *RDF browsers* mais modernos à leveza dos primeiros *RDF browsers* baseados em texto. O código fonte da aplicação é disponibilizado de maneira aberta através do repositório presente no link<sup>3</sup>. A ferramenta foi desenvolvida com o uso da linguagem python, utilizando o *framework Flask*<sup>4</sup> para prover um serviço *Web* leve, deste modo podendo ser hospedado em qualquer *host* dotado com um interpretador python. Além disso, a ferramenta pode ser acessada a partir de qualquer dispositivo com acesso à rede através de um *Web browser*.

LiRB atua como uma interface sobre *endpoints SPARQL*, realizando apenas consultas em SPARQL padrão para acessar os dados, desta maneira podendo ser facilmente adotado em qualquer projeto como um *RDF browser* para qualquer KG RDF. No link<sup>5</sup> encontra-se disponível um KG RDF simples que será utilizado como demonstração da ferramenta. Para acessar a demonstração, basta carregar o KG de demo um qualquer *triplestore* e configurar o arquivo de configurações da ferramenta como indicado no repositório do projeto. Neste artigo foi considerado o *triplestore* GraphDB<sup>6</sup>. A seguir são apresentados os principais modos de visualização de KGs RDF por LiRB.

Figura 14 – Tela de seleção de classes.

The image displays two screenshots of a web application interface for class selection.

**Top Screenshot (L1):** Titled "Highlight Classes L1", it shows two class cards. The left card is for "Company" with URI <http://www.example.lirb.com/Company> and description "a commercial business.". The right card is for "Person P1" with URI <http://www.example.lirb.com/Person> and description "A human being.". The "Person P1" card is highlighted with a red border. Its sub-classes, "Female" and "Male", are listed below the description. Red boxes and labels P1, P2, P3, and P4 indicate the title, URI, description, and sub-classes list respectively.

**Bottom Screenshot (L2):** Titled "Classes list L2", it shows a list of three class cards: "Animal" (URI: <http://www.example.lirb.com/Animal>, description: "A living organism that is not a human being."), "Company" (URI: <http://www.example.lirb.com/Company>, description: "a commercial business."), and "Dog" (URI: <http://www.example.lirb.com/Dog>, description: "A domesticated carnivorous mammal that is kept as a pet or used for various working purposes."). Each card has an "Explore" button.

Fonte: Próprio autor.

### 5.1.1 Navegação de Recursos

A tela inicial apresentada ao acessar a ferramenta é a tela de seleção de classes (Figura 14). Nesta tela são apresentadas as classes existentes no grafo acompanhadas de seus rótulos (P1), URIs (P2), descrições (P3) e sua lista de sub-classes (P4). Nesta tela as classes são dispostas em duas listas: (L1) a lista de classes em destaques, onde são apresentadas as classes marcadas com o atributo “*lirb:has\_spotlight* “true” ” ou através da definição da lista no arquivo de configurações. (L2) É a lista geral de classes, onde são apresentadas todas as classes definidas no KG.

Após selecionar uma classe (Ao clicar em seu respectivo botão “Explore”), é apresentada a lista de recursos pertencentes à classe selecionada. Em qualquer uma das duas telas é

<sup>3</sup> <https://github.com/CaioViktor/LiRB>

<sup>4</sup> <https://flask.palletsprojects.com/en/2.3.x/>

<sup>5</sup> [https://github.com/CaioViktor/LiRB/blob/main/ontology\\_example.ttl](https://github.com/CaioViktor/LiRB/blob/main/ontology_example.ttl)

<sup>6</sup> <https://graphdb.ontotext.com/>

Figura 15 – Tela de navegação de recurso.

The screenshot shows the 'John Doe' resource page in LiRB. At the top, there is a navigation bar with three buttons: 'Visual Graph' (highlighted in red) and 'Timeline' (highlighted in orange). Below the navigation bar, the resource name 'John Doe' is displayed next to a black silhouette icon. Underneath, there is a list of properties: 'A 25 years old software engineering' and '@http://www.example.lirb.com/JohnDoe'. The 'Types' section shows two buttons: 'Living Being' and 'Person'. Below this, there are two main sections: 'Outgoing Properties' and 'Ingoing Properties'. The 'Outgoing Properties' section includes:
 

- Has Age (1) →**: 25
- Has Father (1) →**: <http://www.example.lirb.com/Tom>
- Knows (3) →**:
  - <http://www.example.lirb.com/Tom>
  - <http://www.example.lirb.com/Jane>
  - Jack Smith
- Works At (1) →**:
  - [Software Maker SA.](#)
    - Started At:** 2021-01-01
    - Position In The Company:** Software Engineering

 The 'Ingoing Properties' section includes:
 

- ← Has Member (1)**: <http://www.example.lirb.com/SmithFamily>

Fonte: Próprio autor.

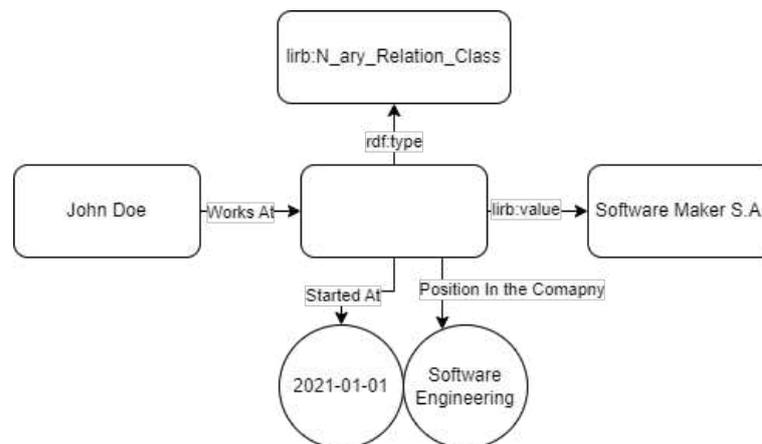
possível pesquisar recursos pela *rdfs:label* ou por trechos da URI, no entanto, quando a busca é realizada na tela de lista de recursos de uma classe específica, a busca é limitada para recursos da classe selecionada. Em KGs muito grandes ou virtuais é recomendável que a T-Box (esquema) e a A-Box (recursos ou instâncias) estejam armazenados em repositórios diferentes. Além disso, nestes casos é recomendável configurar o parâmetro de configuração ‘*USE\_LABELS*’ como ‘False’.

Ao selecionar um recurso da tela de lista de recursos, LiRB apresenta a tela de navegação de recursos (Figura 15). Nesta tela são apresentadas as propriedades com as quais o recurso está envolvido no KG, seja como sujeito de triplas (*outgoing properties*, ou propriedades

partido do recurso atual) ou como objeto de uma tripla (*ingoing properties*, ou propriedades chegando no recurso atual). No caso de propriedades *owl:ObjectProperty* (relacionamentos entre dois recursos), a interface permite a navegação iterativa entre os recursos (apresentados como *links html*) envolvidos nestas. A Figura 15 apresenta as propriedades do recurso “John Doe”, que por sua vez possui a propriedade “Has Father”, relacionando-o ao recurso “Tom”. Ao clicar na URI (ou *rdfs:label* quando disponível) a interface será redirecionada para a página exibindo as propriedades do recurso “Tom”.

Além disso, LiRB oferece a possibilidade de uma fácil integração com *RDF Browsers* visuais, permitindo o redirecionamento para a navegação visual ao clicar no botão destacado em vermelho (Figura 15). Por fim, LiRB também permite que o usuário ative (botão destacado em amarelo na Figura 15) a visualização de propriedades de outras versões do recurso selecionado indicadas por links *owl:sameAs*.

Figura 16 – Representação de como a relação “Works At” é armazenada.



Fonte: Próprio autor.

Além de relações simples representadas por uma única tripla RDF, LiRB também dá o suporte para relações complexas comumente chamadas de *N-ary Relations*. Relacionamentos N-ary (N-ários ou enários) são relacionamentos onde um sujeito está relacionado a mais de um objeto por meio de uma única propriedade ou quando o próprio relacionamento entre dois recursos possui atributos (GIUNTI *et al.*, 2021). Embora atualmente tenha surgido o RDF\* como uma possível solução, esta ainda não é suportada pela maioria dos *triplestores* e demais ferramentas existentes (ARNDT JEEN BROEKSTRA, 2022). Para isto, LiRB optou por seguir uma solução simples e tradicional, o uso de relacionamentos e recursos intermediários<sup>7</sup>.

<sup>7</sup> <https://www.w3.org/TR/swbp-n-aryRelations/>

Na abordagem seguida, a propriedade que representa um relacionamento N-ário relaciona-se com um recurso intermediário sem valor semântico, mas que serve para agregar os n valores complementares do relacionamento. LiRB adota o uso de uma marcação por meio da classe “*lirb:N\_ary\_Relation\_Class*” para abstrair este tipo de relacionamento, apresentando sua semântica correta. A Figura 15 apresenta um relacionamento N-ário na propriedade “*Works At*”, onde o valor da propriedade é apresentado como o recurso “*Software Maker S.A.*”, acompanhado dos metadados da relação (atributos da relação) “*Started At*” e “*Position in The Company*”. A Figura 16 apresenta como a relação “*Works At*” é armazenada na estrutura de triplas RDF.

### 5.1.2 *Timelines de Recursos*

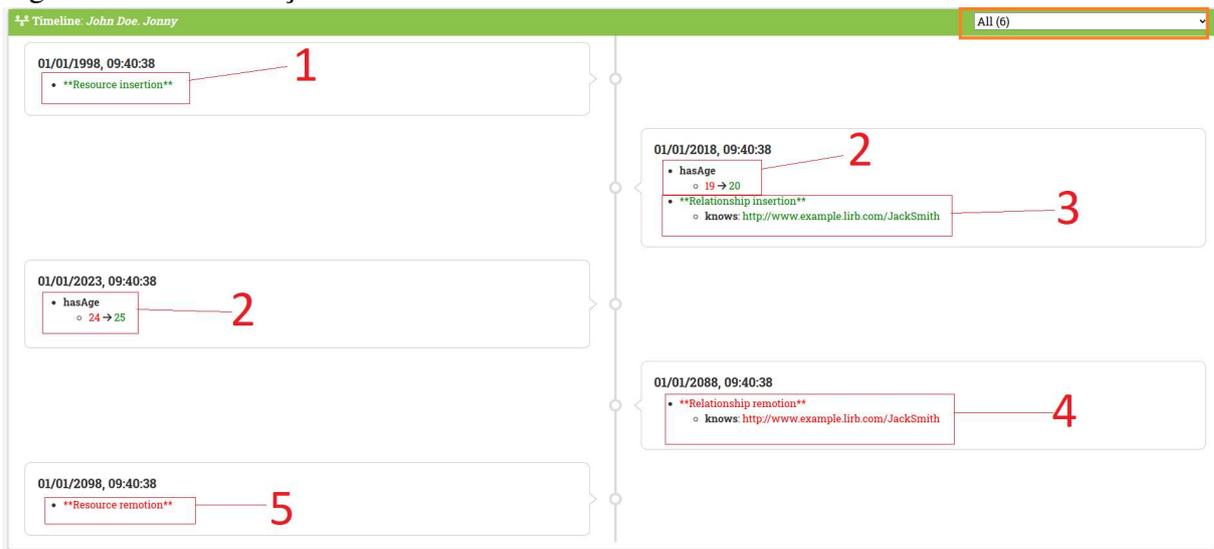
Além da navegação tradicional, LiRB também oferece a possibilidade da visualização de *Timelines* de recursos. A tela de *timeline* de um recurso é acessível através do botão “*Timeline*” na página de navegação de um recurso, posicionada na extremidade direita da barra de título do recurso, quando este possuir a propriedade “*tlo:has\_timeLine*” (acessível através do botão “*Timeline*” destacado em laranja na Figura 15). A tela de exibição de *timelines* (Figura 17) apresenta todos os eventos sofridos pelo recurso selecionado dispostos de maneira agrupada por data e em ordem cronológica. Deste modo, a *timeline* é separada em instantes, onde cada instante possui uma data (no formato dia/mês/ano hora:minutos:segundos) e sua lista de atualizações. Para atualizações de atributos (*owl:DatatypeProperty*) são apresentados os valores antigo e novo atualizados na data (marcação número 2 na Figura 17). Além disso, LiRB também permite filtrar a *timeline* por uma propriedade específica (acessível através da lista na extremidade direita da barra de títulos), apresentando apenas eventos daquele tipo de propriedade (marcação em laranja na Figura 17).

Tabela 2 – Tipos de eventos de timeline

ID	Tipo de evento	Classe do evento
1	Inserção do recurso	tlo:Insertion
2	Atualização de atributo ( <i>owl:DatatypeProperty</i> )	tlo:Update_Property
3	Inserção de relacionamento ( <i>owl:ObjectProperty</i> )	tlo:Insert_Relationship
4	Remoção de relacionamento	tlo:Remove_Relationship
5	Remoção de recurso	tlo:Remotion

Fonte: elaborada pelo autor.

Figura 17 – Visualização de Timeline.



Fonte: Próprio autor.

A anotação da *timeline* de um recurso é realizada seguindo uma ontologia que não será apresentada neste trabalho por conta da limitação de espaço. Deste modo, LiRB dá suporte a 5 diferentes tipos de eventos de *timeline* como apresentado na Tabela 2. O campo *ID* na Tabela 2 representa o identificador do tipo de evento que é utilizado para marcar (numeração vermelha) os exemplos de eventos deste tipo na Figura 17. Tendo como objetivo garantir um bom desempenho da ferramenta, as anotações de *timelines* podem ser armazenadas em um repositório diferente do da A-Box.

### 5.1.3 Consultas Salvas

Além dos modos de visualização anteriormente citados, LiRB também conta com a possibilidade de apresentar consultas previamente salvas. Esta funcionalidade permite que o desenvolvedor salve consultas específicas, para que os resultados destas possa ser visto através da interface pelos usuários.

Figura 18 – Resultado de consulta salva.

### Query Filters

name:

age:  To

birthday:  To

[Filter](#)

---

**Saved Query Results 1 Page 0**

Node	Name	Age	Birthday	Graph
<a href="http://www.example.lirb.com/Jane">http://www.example.lirb.com/Jane</a>	Jane	45	-	<a href="#">View</a>
<a href="http://www.example.lirb.com/JohnDoe">http://www.example.lirb.com/JohnDoe</a>	John Doe	25	1998-01-01	<a href="#">View</a>
<a href="http://www.example.lirb.com/JohnDoe">http://www.example.lirb.com/JohnDoe</a>	Jonny	25	1998-01-01	<a href="#">View</a>
<a href="http://www.example.lirb.com/Tom">http://www.example.lirb.com/Tom</a>	Tom	50	-	<a href="#">View</a>

Fonte: Próprio autor.

Figura 19 – Visualização do recorte do grafo de consulta salva.

## Visual graph

```

graph TD
    Jane((Jane)) -- knows --> JohnDoe((John Doe))
    JohnDoe -- knows --> JackSmith((Jack Smith))
    JohnDoe -- knows --> Tom((Tom))
  
```

[John Doe](#)
en

*A 25 years old software engineering*

John Doe<sup>en</sup> · Jonny

Types:

<http://www.example.lirb.com/Person>

RDF Rank:

0

rdfs:comment  
A 25 years old software engineering<sup>en</sup>

rdfs:label  
John Doe<sup>en</sup> [Show 1 more](#)

<http://www.example.lirb.com/hasAge>  
25 xsd:integer

Fonte: Próprio autor.

A Figura 18 apresenta o resultado de uma consulta salva que recupera todas as pessoas existentes no KG acompanhadas de seu nome, idade e data de nascimento. A interface permite a aplicação de filtros sobre atributos retornados nas consultas salvas de maneira automática, necessitando que o desenvolvedor apenas indique quais campos devem ser filtráveis e o

tipo de filtro a ser aplicado. A interface dá suporte a filtros sobre campos “string”, numéricos e datas. Para campos do tipo “string” são aplicados filtros do tipo contém *substring*. Para campos numéricos e datas são aplicados filtros do tipo faixa, onde para aplicar filtros do tipo igualdade basta colocar limites iguais.

Além disso, LiRB conta com uma integração com a ferramenta de navegação visual do *triplestore* GraphDB, permitindo o redirecionamento para a navegação visual de recursos do KG. Em particular, o desenvolvedor pode escrever consultas que construam recortes do KG, tendo como ponto de partida o valor de variáveis retornadas em consultas salvas, deste modo permitindo a visualização visual das relações destacadas pela respectiva consulta. A Figura 19 apresenta o recorte mostrado ao clicar na opção “View” destacado em vermelho do recurso “JohnDoe”, assim mostrado o recorte do grafo que apresenta todos os conhecidos deste recurso.

## 6 RESULTADOS E DISCUSSÕES

Este capítulo cobre os resultados dos experimentos realizados nesta tese. A seção 6.1 discute os resultados dos experimentos realizados sobre o *dataset Daily-life* definido nesta tese. A seção 6.2 apresenta os resultados obtidos pelo Auto-KGQA no *benchmark* definido em Sequeda et al. (SEQUEDA *et al.*, 2023), avaliando o desempenho do *framework* em perguntas de diferentes complexidades. A Seção 6.3 apresenta os resultados obtidos no *dataset* SciQA (AUER *et al.*, 2023), avaliando o impacto da abordagem *few-shot learning* no desempenho do *framework*. Por fim, a seção 6.4 apresenta algumas discussões sobre os resultados e apresenta as respostas para as Questões de Pesquisa Gerais (QPGs) definidas nesta tese.

### 6.1 Daily-life Dataset

Para esta investigação, um KG RDF curto sobre o domínio da vida cotidiana foi elaborado. O KG conta com uma definição simplificada das classes Seres Vivos, Pessoas, Animais, Empresas, Emprego, etc. A serialização RDF do KG está disponível no link<sup>1</sup>. A Figura 20 apresenta o esquema do KG utilizado nos experimentos. Além disso, cada conjunto de triplas possui comentários em linguagem natural que descreve a semântica destas, atuando assim como a abordagem *Text on Prompt*.

Para avaliar o desempenho do ChatGPT como KGQA foram definidas 15 questões sobre este KG, as quais buscam avaliar a capacidade do ChatGPT de responder desde consultas simples que podem ser respondidas com base em uma única tripla, bem como consultas complexas que necessitam o uso de caminhos de relações e/ou funções de agregação.

As Tabelas 3, 4, 5 apresentam as questões utilizadas para avaliar o ChatGPT como ferramenta de KGQA. Nestas tabelas são apresentados os IDs da questões que serão utilizados para identificá-las, o texto da questão, a consulta SPARQL e o conjunto de resultado que atuará como *gold-standard* para cada questão, além da razão da inclusão de tal consulta.

A Tabela 6 apresenta um resumo dos experimentos realizados neste dataset. As colunas (1) - (4) representam os resultados da execução manual via interface Web do ChatGPT para as seguintes abordagens respectivamente: (1) Passagem completa do KG, solicitando as respostas diretamente em linguagem natural; (2) Passagem completa do KG, solicitando a tradução da pergunta original para um consulta SPARQL que a represente; (3) Passagem de

<sup>1</sup> [https://drive.google.com/file/d/1hhZksmghH4HqgpTRR8703DG4ZjU9izkU/view?usp=drive\\_link](https://drive.google.com/file/d/1hhZksmghH4HqgpTRR8703DG4ZjU9izkU/view?usp=drive_link)

Tabela 3 – Questões utilizadas para avaliar o ChatGPT

ID	Questão	SPARQL	Resposta	Razão
Q <sub>1</sub>	who is John?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT ?wholsJohn ?label WHERE {   ?wholsJohn a lib:Person.   ?wholsJohn rdfs:label ?label.   FILTER(REGEX(?label, "John", "i")) } PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT ?wholsJohn ?label WHERE {   ?wholsJohn a lib:Person.   ?wholsJohn rdfs:label ?label.   FILTER(REGEX(?label, "Tom", "i")) }</pre>	<p>http://www.example.lirb.com/JohnDoe</p> <p>http://www.example.lirb.com/John</p>	Consulta simples para avaliar se a ferramenta consegue associar corretamente um termo da consulta à um recurso. Esta faz um grupo formado pelas consultas 1,2,3 para avaliar se a ferramenta produz variações do mesmo padrão de consulta corretamente
Q <sub>2</sub>	who is Tom?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT ?wholsJohn ?label WHERE {   ?wholsJohn a lib:Person.   ?wholsJohn rdfs:label ?label.   FILTER(REGEX(?label, "Tom", "i")) }</pre>	<p>http://www.example.lirb.com/Tom</p>	Consulta simples para avaliar se a ferramenta consegue associar corretamente um termo da consulta à um recurso. Esta faz um grupo formado pelas consultas 1,2,3 para avaliar se a ferramenta produz variações do mesmo padrão de consulta corretamente
Q <sub>3</sub>	who is Buddy?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT ?wholsJohn ?label WHERE {   ?wholsJohn rdfs:label ?label.   FILTER(REGEX(?label, "Buddy", "i")) }</pre>	<p>http://www.example.lirb.com/Buddy</p>	Consulta simples para avaliar se a ferramenta consegue associar corretamente um termo da consulta à um recurso. Esta faz um grupo formado pelas consultas 1,2,3 para avaliar se a ferramenta produz variações do mesmo padrão de consulta corretamente
Q <sub>4</sub>	How old is John Doe?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT ?wholsJohn ?age WHERE {   ?wholsJohn a lib:Person.   ?wholsJohn rdfs:label ?label;   lib:hasAge ?age   FILTER(REGEX(?label, "John", "i")) }</pre>	<p>http://www.example.lirb.com/JohnDoe</p> <p>"25^^xsd:integer</p> <p>http://www.example.lirb.com/John</p> <p>"25^^xsd:integer</p>	Consulta simples que busca um atributo específico de um recurso específico
Q <sub>5</sub>	who John Doe Knows?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lirb.com/&gt; SELECT DISTINCT ?p WHERE {   ?wholsJohn a lib:Person.   ?wholsJohn rdfs:label ?label;   lib:knows ?p   FILTER(REGEX(?label, "John", "i")) }</pre>	<p>http://www.example.lirb.com/Tom</p> <p>http://www.example.lirb.com/Iane</p> <p>http://www.example.lirb.com/Jacksmith</p> <p>http://www.example.lirb.com/SomeOne</p>	Consulta simples que busca triplas para um relacionamento específico de um recurso específico

Fonte: elaborada pelo autor.

Tabela 4 – Questões utilizadas para avaliar o ChatGPT

ID	Questão	SPARQL	Resposta	Razão
Q <sub>6</sub>	who is the oldest person?	<pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lib.com/&gt; SELECT ?p ?age WHERE { ?p a lib:Person; lib:hasAge ?age } ORDER BY DESC(?age) LIMIT 1 </pre>	<pre> http://www.example.lib.com/Tom "50^^xsd:integer </pre>	Consulta de agregação que utiliza termo não presente diretamente no esquema do KG, necessitando correta interpretação
Q <sub>7</sub>	who is the youngest person?	<pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lib.com/&gt; SELECT ?p ?age WHERE { ?p a lib:Person; lib:hasAge ?age } ORDER BY ASC(?age) LIMIT 1 </pre>	<pre> http://www.example.lib.com/JohnDoe "25^^xsd:integer </pre>	Consulta de agregação que utiliza termo não presente diretamente no esquema do KG, necessitando correta interpretação
Q <sub>8</sub>	how many persons exist?	<pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lib.com/&gt; SELECT ( COUNT(?p) as ?qid) WHERE { ?p a lib:Person; } </pre>	<pre> "6^^xsd:integer </pre>	Consulta de agregação que utiliza termo não presente diretamente no esquema do KG, necessitando correta interpretação
Q <sub>9</sub>	what is the average age of people?	<pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX lib: &lt;http://www.example.lib.com/&gt; SELECT (AVG(?age) as ?media) WHERE { ?p a lib:Person; lib:hasAge ?age } </pre>	<pre> 40* </pre>	Consulta de agregação que utiliza termo não presente diretamente no esquema do KG, necessitando correta interpretação

Fonte: elaborada pelo autor.

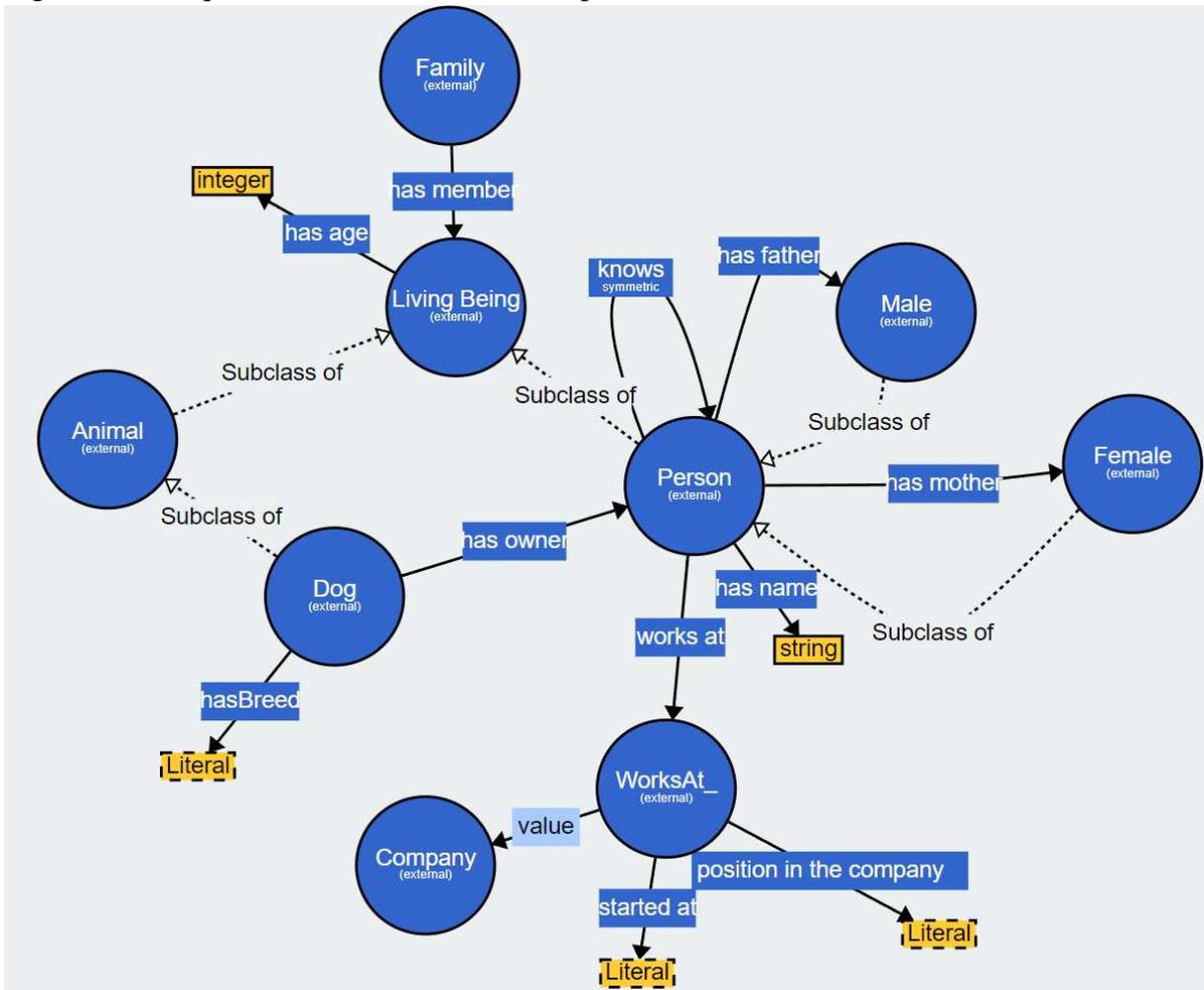
Legenda: \* Ao considerar com inferências ligadas o resultado por ser 36.25.

Tabela 5 – Questões utilizadas para avaliar o ChatGPT

ID	Questão	SPARQL	Resposta	Razão
Q10	who is over 30?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX irb: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX ?p ?age WHERE {   ?p a irb:Person;   irb:hasAge ?age   FILTER(?age &gt; 30) }</pre>	<p>http://www.example.irb.com/jane</p> <p>"45"^^xsd:integer</p> <p>http://www.example.irb.com/Tom</p> <p>"50"^^xsd:integer</p>	Consulta utilizando filtro sobre atributo específico, no entanto, o atributo é implícito
Q11	what is the relation between Jane and Tom?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX irb: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; SELECT ?j ?p ?t WHERE {   ?j rdfs:label ?j_1   ?t rdfs:label ?t_1   FILTER(REGEX(?j_1, "Jane", "I"))   FILTER(REGEX(?t_1, "Tom", "I")) }</pre>	<p>http://www.example.irb.com/jane</p> <p>http://www.example.irb.com/knows</p> <p>http://www.example.irb.com/Tom</p>	Consulta buscando encontrar a identificação do relacionamento entre dois recursos específicos
Q12	Does John Doe have a job? if so, where does he work, his position and when did he start?	<pre>PREFIX irb: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt; PREFIX dt: &lt;http://purl.org/NET/44dm/timeline.owl#&gt; BASE &lt;http://www.example.irb.com/&gt; PREFIX irb: &lt;https://raw.githubusercontent.com/CaioViktor/LRB/main/irb_ontology.ttl&gt; SELECT ?company ?position ?startDate WHERE {   ?john rdf:type &lt;Person&gt;;   ?john rdfs:label "John Doe"@en .   ?john &lt;worksAt&gt; ?job .   ?job &lt;positionInCompany&gt; ?position ;   &lt;startedAt&gt; ?startDate ;   irb:value ?company . }</pre>	<p>http://www.example.irb.com/SoftwareMaker</p> <p>"Software Engineering"</p> <p>"2021-01-01T00:00:00.873-03:00"^^xsd:datetime</p>	Consulta complexa composta por múltiplas consultas simples, também envolvendo a necessidade da abstração de uma classe representando um relacionamento
Q13	who is the father of buddy's owner?	<pre>PREFIX irb: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; BASE &lt;http://www.example.irb.com/&gt; SELECT ?father ?fatherName WHERE {   ?buddy rdf:type &lt;Dog&gt;;   rdfs:label "Buddy".   ?buddy &lt;hasOwner&gt; ?owner .   ?owner &lt;hasFather&gt; ?father .   ?father &lt;hasName&gt; ?fatherName . }</pre>	<p>http://www.example.irb.com/Tom</p> <p>"Tom"</p>	consulta complexa que depende da identificação de recursos intermédiairos utilizando um relacionamento com um recurso específico como filtro
Q14	how old is the father of the person with 25 years old?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; BASE &lt;http://www.example.irb.com/&gt; SELECT ?father ?fatherAge WHERE {   ?person rdf:type &lt;Person&gt;;   &lt;hasAge&gt; 25   ?person &lt;hasFather&gt; ?father .   ?father &lt;hasAge&gt; ?fatherAge . }</pre>	<p>http://www.example.irb.com/Tom</p> <p>"50"^^xsd:integer</p>	consulta complexa que depende da identificação de recursos intermédiairos utilizando uma atributo específico como filtro
Q15	who is Barack Obama?	<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX irb: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; SELECT ?wholsJohn ?label WHERE {   ?wholsJohn a irb:Person.   ?wholsJohn rdfs:label ?label   FILTER(REGEX(?label, "Barack Obama", "I")) }</pre>	<p>consulta que não deve ser respondida pelo grafo</p>	

Fonte: elaborada pelo autor.

Figura 20 – Esquema do KG utilizado nos experimentos.



Fonte: Próprio autor.

apenas triplas da T-Box do KG, solicitando a tradução da pergunta original para um consulta SPARQL que a represente; e (4) Passagem de apenas triplas da A-Box do KG, solicitando a tradução da pergunta original para um consulta SPARQL que a represente. Por sua vez, as colunas (5) - (7) apresentam os resultados alcançados na abordagem apresentada na Seção 4 utilizando os modelos GTP-3.5 Turbo, GTP-4 e GPT-4 Turbo respectivamente. Por fim, a coluna (8) apresenta os resultados dos experimentos ao utilizar a abordagem de RAG presente no *framework langchain*.

Tabela 6 – Resumo de resultados de experimentos.

Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
$Q_1$	1	1	1	0	1	1	1	1
$Q_2$	0	1	1	0	1	1	1	1
$Q_3$	1	1	1	0	1	1	1	0
$Q_4$	1	1	1	0	1	1	1	1
$Q_5$	1	1	1	0	1	1	1	1
$Q_6$	1	1	1	0	1	1	1	1
$Q_7$	1	1	0	0	1	1	1	0
$Q_8$	0	1	0	0	1	1	1	1
$Q_9$	1	1	1	0	1	1	0	0
$Q_{10}$	0	1	1	0	0	0	0	1
$Q_{11}$	0	1	1	0	0	1	1	0
$Q_{12}$	1	0	0	0	1	1	0	0
$Q_{13}$	1	1	0	0	1	1	1	1
$Q_{14}$	0	0	1	0	1	1	1	1
$Q_{15}$	1	1	1	1	1	1	1	1
<b>#Correct</b>	10	13	11	1	13	<b>14</b>	12	10
<b>Acurácia</b>	67%	87%	73%	6,7%	87%	<b>93%</b>	80%	67%

Fonte: elaborada pelo autor.

Nota: (1) NL; (2) T-Box + A-Box; (3) T-Box; (4) A-Box; (5) Auto-KGQA (GTP-3.5 Turbo); (6) Auto-KGQA (GTP-4);

(7) Auto-KGQA (GTP-4 Turbo); (8) Langchain RAG (GTP-3.5 Turbo).

### 6.1.1 Experimentos Manuais

Experimentos foram executados manualmente diretamente na interface Web do ChatGPT disponível no link<sup>2</sup>. Estes experimentos objetivaram avaliar a capacidade da ferramenta de atuar na tarefa de KGQA, além de ajudar a decidir qual abordagem pode obter os melhores resultados (1) Geração da resposta diretamente em linguagem natural e (2) Tradução da pergunta para uma consulta SPARQL. Dentro da segunda abordagem, foram também avaliados cenários onde são passadas (2.a) triplas da T-Box + A-Box (totalizando o KG completo); (2.b) triplas apenas da T-Box; (2.c) triplas apenas da A-Box.

Nestes experimentos o KG de entrada foi dado uma única vez no início da interação, sendo passado em sua serialização *Turtle RDF* para o ChatGPT. Todas as questões foram realizadas dentro de uma única instância de chat isolada de demais conversas. Cada questão foi passada para o ChatGPT estruturada como o seguinte *prompt*:

#### 1. Geração direta da resposta em linguagem natural:

Write a SPARQL query for the question using only classes and properties defined in the graph and remember to include the base uri declaration in

<sup>2</sup> <https://chatgpt.com/>

the query and non-essential properties as optional if needed. Declare filters on strings as filters operations over regex function using the case insensitive flag. The question is the following:  
 {question}

## 2. Tradução para consulta SPARQL intermediária:

### a) T-Box+A-Box:

Write a SPARQL query for the question using only classes and properties defined in the graph and remember to include the base uri declaration in the query and non-essential properties as optional:  
 {question}

### b) T-Box:

Write a SPARQL query for the question using only classes and properties defined in the graph and remember to include the base uri declaration in the query and non-essential properties as optional if needed. Declare filters on strings as filters operations over regex function using the case insensitive flag. The question is the following:  
 {question}

### c) A-Box:

Write a SPARQL query for the question using only classes and properties defined in the graph and remember to include the base uri declaration in the query and non-essential properties as optional if needed. Declare filters on strings as filters operations over regex function using the case

insensitive flag. The question is the following:  
 {question}

A seguir estão disponíveis os links para os logs das execuções dos experimentos:

1. log textual<sup>3</sup>, imagens<sup>4</sup>;
2. a) log textual<sup>5</sup>, imagens<sup>6</sup>;
- b) log textual<sup>7</sup>, imagens<sup>8</sup>;
- c) log textual<sup>9</sup>, imagens<sup>10</sup>.

Com base nestes experimentos é possível observar que a abordagem da tradução da questão original para uma consulta SPARQL foi mais eficiente do que a geração direta de respostas pelo LLM envolvendo apenas a linguagem natural, onde a abordagem direta acertou 10 das 15 questões perguntadas, alcançando uma taxa de acerto de 67%, enquanto a melhor abordagem envolvendo a tradução SPARQL alcançou uma taxa de acerto de 87%. Ao observar individualmente as respostas dadas pelo ChatGPT no cenário utilizando respostas diretas<sup>11</sup> é possível notar que embora em muitos casos o LLM apresente o raciocínio correto para responder a questão, a conclusão alcançada está incorreta. Isto torna-se evidente principalmente em consultas que envolvam cálculos matemáticos ou comparações. A seguir segue um exemplo de uma resposta incorreta dada pelo ChatGPT que apresenta um raciocínio correto, mas conclusão errada:

```

1
2 User: Taking in account the graph, who is over 30?
3
4 ChatGPT: Based on the provided RDF graph, the individuals
           who are over 30 years old can be determined by examining
           their ages. Here are the relevant portions of the graph
           :

```

<sup>3</sup> [https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive\\_link](https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive_link)

<sup>4</sup> [https://drive.google.com/drive/folders/1C7qDOo3HEYvHs\\_65THklN5ujjBJeUzP5?usp=drive\\_link](https://drive.google.com/drive/folders/1C7qDOo3HEYvHs_65THklN5ujjBJeUzP5?usp=drive_link)

<sup>5</sup> [https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive\\_link](https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive_link)

<sup>6</sup> [https://drive.google.com/drive/folders/141CGMj88k59ZsZvu1-1cV1x2deEGqJEU?usp=drive\\_link](https://drive.google.com/drive/folders/141CGMj88k59ZsZvu1-1cV1x2deEGqJEU?usp=drive_link)

<sup>7</sup> [https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive\\_link](https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive_link)

<sup>8</sup> [https://drive.google.com/drive/folders/1q0CzjNv7pSnvkxCdrRMSVNo8kgCl0e9V?usp=drive\\_link](https://drive.google.com/drive/folders/1q0CzjNv7pSnvkxCdrRMSVNo8kgCl0e9V?usp=drive_link)

<sup>9</sup> [https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive\\_link](https://drive.google.com/file/d/1ZOfmdceiwXOQUwpQK7eHOQOKRpjRcHvI/view?usp=drive_link)

<sup>10</sup> [https://drive.google.com/drive/folders/1WyOZXgRRJmo9Ewwmd-EML60oqsoEhRni?usp=drive\\_link](https://drive.google.com/drive/folders/1WyOZXgRRJmo9Ewwmd-EML60oqsoEhRni?usp=drive_link)

<sup>11</sup> <https://docs.google.com/spreadsheets/d/1fiFOk38y5rxNBH6gZ-M-etXAynj5GuZT/edit?usp=sharing&oid=100005366083907906610&rtpof=true&sd=true#gid=333227843>

```

5
6 <JohnDoe> <hasAge> "25"^^xsd:integer .
7 <Tom> <hasAge> "50"^^xsd:integer .
8 <Jane> <hasAge> "45"^^xsd:integer .
9
10 Based on the ages provided: "<JohnDoe>" is 25 years old.
    "<Tom>" is 50 years old. "<Jane>" is 45 years old. Only
    "<Tom>" is over 30 years old (specifically, he is over
    30 and over 30 + 30 = 60). Therefore, "<Tom>" is the
    individual who is over 30 in the graph.

```

Na resposta dada a cima pelo ChatGPT é possível notar que o LLM selecionou o conjunto de triplas corretas necessárias para gerar a resposta, inclusive interpretando-as corretamente, no entanto, no decorrer da resposta o LLM gera alucinações que não fazem sentido lógico, gerando uma conclusão falsa. Além disso, a abordagem de geração direta das respostas conta com mais uma desvantagem quando considerando respostas que necessitem a análise ou comparação de um grande conjunto de dados considerando que o LLM tem uma entrada de tamanho limitado. Por exemplo, a consulta anteriormente apresenta para retornar as pessoas com idade maior que 30 anos não seria possível de ser executada em KGs com uma grande quantidade de instâncias. Seguindo este raciocínio, respostas que envolvessem longas listas também não seriam possíveis.

Deste modo, a abordagem de geração da tradução SPARQL da consulta original e sua posterior execução pelo *triplestore* hospedando o KG demonstra-se mais adequada em cenários reais com KGs massivos, pelos seguintes pontos:

- Evita alucinações durante o processo de raciocínio da consulta;
- Permite a avaliação de grandes conjuntos de instâncias;
- Permite a saída de resultados com grandes listas de resultados;
- Permite a potencial omissão de trechos do KG para o LLM.

Ao considerar os cenários envolvendo a tradução da consulta para SPARQL é possível notar a relevância da passagem de elementos de ambas T-Box e A-Box, isto é notável ao observar que na abordagem onde as duas são passadas, o ChatGPT conseguiu gerar consultas SPARQL que retornavam a resposta correta para 13 das 15 questões, alcançado uma taxa de 87% de acerto.

Ao analisar as respostas dadas pelo ChatGPT<sup>12</sup> é possível notar que o LLM conseguiu gerar consultas SPARQL que até mesmo precisavam utilizar propriedades não definidas na T-Box, utilizando fragmentos da A-Box para escrever a consulta.

Por sua vez, ao considerar apenas a passagem individual da T-Box, o ChatGPT acertou 11 das 15 (73%). Ao analisar as respostas dadas pelo ChatGPT<sup>13</sup> é possível observar que a principal fonte de erros está no fato da T-Box não definir de maneira explícita todas as propriedades utilizadas ou convenções seguidas na A-Box. Este cenário se aproxima de cenários de mundo real, onde a T-Box não é capaz de modelar exaustivamente todo o domínio sendo abordado, seja por conta de reuso de conceitos de outras ontologias ou por conta do conhecimento incompleto durante a modelagem da T-Box.

Deste modo, fica clara a importância de triplas da A-Box para a geração de consultas SPARQL por LLMs, principalmente considerando que estas podem ser facilmente utilizadas na escrita da consulta, ao simplesmente substituir elementos da tripla por variáveis na consulta a ser gerada.

Contudo ao considerar apenas a passagem da A-Box apenas uma questão de 15 foi acertada. Ao analisar as respostas dadas pelo ChatGPT<sup>14</sup> é possível observar que a principal fonte de erros foi o uso incorreto de URIs de propriedades, utilizando prefixos incorretamente, embora a estrutura da consulta estivesse correta.

### **6.1.2 Experimentos com Auto-KGQA**

Tendo em vista as observações dos experimentos manuais, Auto-KGQA foi feito tendo como objetivo diminuir o tamanho do KG passado como entrada para o LLM, considerando tanto T-Box, quanto A-Box. Considerando o uso do modelo GPT-3.5 Turbo, Auto-KGQA gerou 13 consultas SPARQL contendo as respostas corretas para as 15 perguntas definidas, empatando com os melhores resultados obtidos nos experimentos manuais. Auto-KGQA com GPT-4 gerou consultas SPARQL que traduziram corretamente 14 questões das 15 testadas, destacando-se como o melhor modelo, com uma precisão de 93% no conjunto de dados. Auto-KGQA com GPT-4 Turbo gerou consultas SPARQL que traduziram corretamente 12 questões das 15 testadas,

<sup>12</sup> <https://docs.google.com/spreadsheets/d/1fiFOk38y5rxNBH6gZ-M-etXAynj5GuZT/edit?usp=sharing&oid=100005366083907906610&rtpof=true&sd=true#gid=1655314002>

<sup>13</sup> <https://docs.google.com/spreadsheets/d/1fiFOk38y5rxNBH6gZ-M-etXAynj5GuZT/edit?usp=sharing&oid=100005366083907906610&rtpof=true&sd=true#gid=248910128>

<sup>14</sup> <https://docs.google.com/spreadsheets/d/1fiFOk38y5rxNBH6gZ-M-etXAynj5GuZT/edit?usp=sharing&oid=100005366083907906610&rtpof=true&sd=true#gid=2133330611>

obtendo um resultado pior que o GPT-3.5 Turbo. Vale destacar que a versão GPT-4 Turbo ainda é uma versão beta do modelo. Uma análise das respostas LN geradas usando o GPT-4 Turbo indica que as respostas foram muito mais complexas e detalhadas do que aquelas obtidas com o GPT-3.5 Turbo.

Ao avaliar os resultados gerados pelo Auto-KGQA de perto, é possível observar que as principais fontes de erros foi o uso incorreto de *namespaces* de propriedades e a ausência de cláusulas *OPTIONAL* em propriedades opcionais. Além disso, a consulta de ID 10 foi errada por todos os modelos avaliados no *framework*. Isto se deve por conta da questão não contar explicitamente como nenhuma referência a elementos do KG, resultando na passagem de um recorte vazio do KG para o LLM. No entanto, ao avaliar esta questão nos cenários manuais, onde o contexto do diálogo anterior foi considerado, o ChatGPT conseguiu inferir que a pergunta sendo feita fazia referência a idade de pessoas. Esta foi uma questão propositalmente escrita como uma questão incompleta no formato de anáfora, onde uma pergunta incompleta necessita do histórico da conversa para ser interpretada. Ao realizar a reescrita da consulta, completando-a, esta ficaria da seguinte maneira: “Quem são as pessoas com idade maior que 30 anos?”. Todos os três modelos conseguiram traduzi-la corretamente. Considerando esta variação da consulta, o GPT-4 gerou a melhor resposta, em termos de simplicidade e completude.

### 6.1.3 Experimentos com Langchain

Para comparar o desempenho de Auto-KGQA com outras abordagens de RAG, experimentos foram realizados com o RAG do *framework Langchain*<sup>15</sup> (correspondendo a Coluna (8)) e com *Langchain GraphSparqlQAChain*<sup>16</sup>.

Nestes experimentos as triplas do KG foram indexadas pelo RAG do *Langchain*, onde cada tripla foi inserida como um documento, onde em tempo de execução são buscados os top 90 documentos (triplas) relevantes para a questão de entrada. Estes documentos são então convertidos para Turtle e passados como o fragmento do KG para o LLM que gera apenas uma única tradução de consulta SPARQL. Foram escolhidos 90 documentos, pois este foi o número que obteve os melhores resultados sem ultrapassar o número de *tokens* médios utilizados na abordagem Auto-KGQA. Esta abordagem utilizou o modelo GPT-3.5 Turbo e gerou 10 respostas corretas. Deste modo, todas as versões de Auto-KGQA conseguiram um melhor desempenho,

<sup>15</sup> [https://python.langchain.com/docs/expression\\_language/cookbook/retrieval](https://python.langchain.com/docs/expression_language/cookbook/retrieval)

<sup>16</sup> [https://python.langchain.com/docs/use\\_cases/graph/graph\\_sparql\\_qa](https://python.langchain.com/docs/use_cases/graph/graph_sparql_qa)

embora tenham usado um número maior de *tokens* no subgrafo passado ao LLM.

Os resultados dos experimentos com Langchain GraphSparqlQACHain<sup>17</sup> foram muito ruins e não foram incluídos na tabela 6.

#### 6.1.4 Redução no número de Tokens passados

Ao considerar o tamanho do KG passado como entrada para o LLM pelo Auto-KGQA é possível observar uma redução de 81.98% no número de *tokens* ao considerar o KG completo sem inferências. Neste cenário o número médio de *tokens* do subgrafo do KG passado para o LLM é 720.42, enquanto o número de *tokens* do KG sem inferências é 2977. Ao considerar o KG acrescido das inferências geradas ao aplicar o *reasoner* OWL 2 QL<sup>18</sup> seu tamanho em *tokens* é de 14879, o que o torna impossível de ser passado para a API do ChatGPT, mostrando a necessidade da abordagem de filtragem adotada por Auto-KGQA que gerou uma diminuição de 95.23% no número de *tokens*. A Tabela 7 apresenta a quantidade de *tokens* passados pelo Auto-KGQA por questão para a API do ChatGPT.

Tabela 7 – Quantidade de tokens passados pelo Auto-KGQA por questão.

ID	tokens
Q <sub>1</sub>	296
Q <sub>2</sub>	794
Q <sub>3</sub>	337
Q <sub>4</sub>	891
Q <sub>5</sub>	1123
Q <sub>6</sub>	968
Q <sub>7</sub>	828
Q <sub>8</sub>	751
Q <sub>9</sub>	686
Q <sub>10</sub>	708
Q <sub>11</sub>	644
Q <sub>12</sub>	574
Q <sub>13</sub>	438
Q <sub>14</sub>	1048
Q <sub>15</sub>	965

Fonte: elaborada pelo autor.

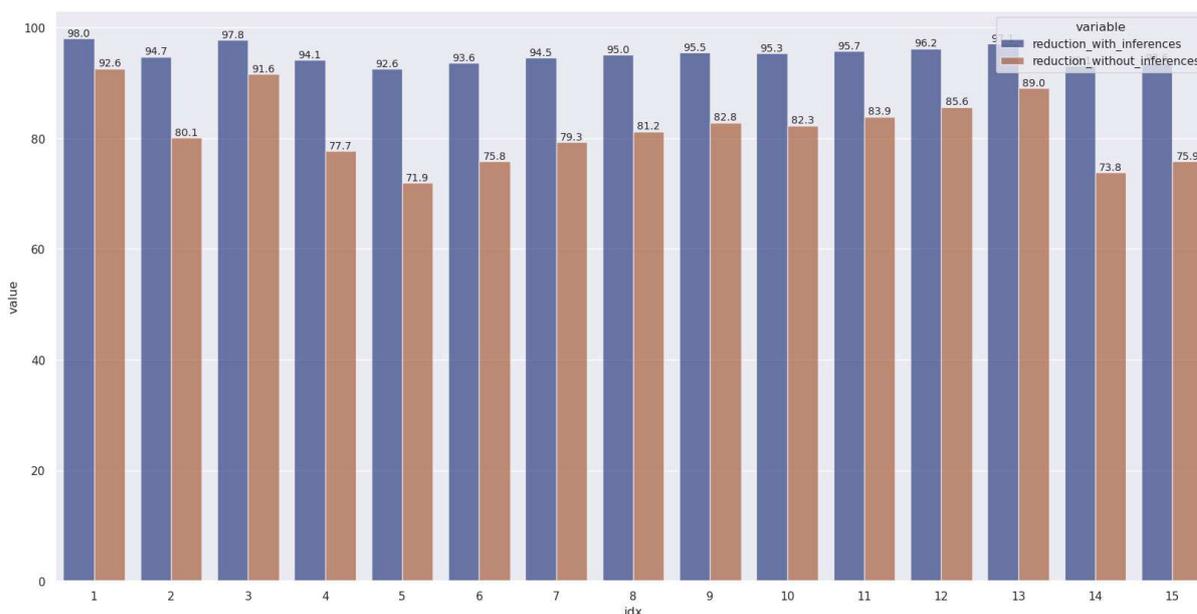
A abordagem utilizando o RAG do *Langchain* utilizou uma média de 1531,6 *tokens*, mostrando que Auto-KGQA conseguiu reduzir o número de *tokens* do KG passados como entrada de maneira eficaz e eficiente, alcançando uma melhor acurácia com uma menor quantidade de *tokens*.

<sup>17</sup> [https://python.langchain.com/docs/use\\_cases/graph/graph\\_sparql\\_qa](https://python.langchain.com/docs/use_cases/graph/graph_sparql_qa)

<sup>18</sup> [urlhttps://www.w3.org/TR/owl2-profiles/#OWL\\_2\\_QL\\_2](https://www.w3.org/TR/owl2-profiles/#OWL_2_QL_2)

A Figura 21 apresenta a redução percentual no número de *tokens* para cada questão ao considerar o KG com inferências (azul) e o KG sem inferências (laranja). Com base na figura é possível constatar que todas as questões tiveram uma redução consistente de mais do que 70% na quantidade de *tokens* quando não considerando as inferências. Ao considerar as inferências, a redução se mostra também consistentemente a cima de 92%.

Figura 21 – Redução percentual no número de *tokens* passados por questão comparado ao KG com e sem inferências.



Fonte: Próprio autor.

## 6.2 Resultados do Auto-KGQA em um *benchmark* de questões empresariais em linguagem natural

Esta seção utiliza o *benchmark* proposto em (SEQUEDA *et al.*, 2023) para comparar o desempenho de Auto-KGQA contra outras soluções propostas na comunidade em questões complexas sobre domínios específicos. Os autores definem uma camada semântica sobre um banco de dados relacional como um KG RDF neste *benchmark*. Os resultados podem ser encontrados em<sup>19</sup>. O *benchmark* é composto por 43 questões divididas em quatro categorias:

- *Low Question/Low Schema* (LL): Estas são consultas simples de seleção de atributos que usam um pequeno número de tabelas (ou seja, 0 - 4) de um esquema desnormalizado.
- *High Question/Low Schema* (HL): Essas consultas realizam operações complexas, como funções de agregação ou que dependem de conhecimento específico do domínio para sua

<sup>19</sup> <https://bit.ly/3JEp242>

interpretação. Elas também usam um pequeno número de tabelas (ou seja, 0 a 4) de um esquema desnormalizado.

- *Low Question/High Schema* (LH): São consultas simples de seleção de atributos que utilizam um número maior de tabelas ( $> 4$ ) de um esquema normalizado, com tabelas de junção muitos para muitos, etc.
- *High Question/High Schema* (HH): Essas consultas realizam operações complexas, como funções de agregação ou que dependem de conhecimento específico do domínio para sua interpretação. Eles usam um número maior de tabelas ( $> 4$ ) de um esquema normalizado, com tabelas de junção muitos para muitos, etc.

Os experimentos utilizaram o modelo GPT-4, assim como o utilizado pelos autores no trabalho original. Além disso, durante a fase de extração do subgrafo relevante (Algoritmo 4) a profundidade da vizinhança extraída (parâmetro *edge\_hops*) foi configurada para 4 e não foi limitado o número de visitas por tipo de propriedade (parâmetro *max\_edges\_by\_property*). Por fim, o *threshold* de relevância mínima (*min\_relevance\_threshold*) para triplas utilizado no Algoritmo 7 foi configurado para 0.45 e o número máximo de resultados (*maximun\_number\_results*) retornados foi configurado para taxa de 90% das triplas inicial selecionadas.

A métrica de avaliação seguida em (SEQUEDA *et al.*, 2023) considera uma respostas como correta quando esta é igual ao conjunto de resposta retornado pela consulta do *golden-standard*, desconsiderando a ordem das linhas e colunas (além do nome destas) do resultado. Deste modo, a avaliação dos resultados foi realiza em duas etapas, na primeira um *script* automatizado converteu os resultados das consultas (retornadas pelo *framework* e do *golden-standard*) para um conjunto de conjuntos, onde o primeiro conjunto representa a lista de linhas da resposta. Por sua vez, os conjuntos mais internos representam os valores de uma linha da resposta, desta maneira desconsiderando a ordem de linhas e colunas. Nesta etapa automatizada, uma consulta é considerada como correta quando os dois conjuntos são iguais. Na segunda etapa foi realizada uma análise manual das consultas consideradas como incorretas na etapa anterior. Nesta etapa, consultas que apresentavam campos adicionais que representavam apenas as URIs dos recursos selecionados foram consideradas como corretas, visto que esta é uma característica desejável no *framework* (embora apenas duas consultas tenham se enquadrado neste caso).

A tabela 8 mostra os resultados obtidos com Auto-KGQA, comparado com os resultados obtidos em (SEQUEDA *et al.*, 2023). Auto-KGQA obteve resultados melhores que os relatados em (SEQUEDA *et al.*, 2023), não apenas para o conjunto de todas as consultas,

mas também para cada categoria de perguntas. Auto-KGQA obteve uma precisão de 67,4% no conjunto geral de perguntas, alcançando uma melhoria de 13,2% em relação a (SEQUEDA *et al.*, 2023).

Tabela 8 – Comparação entre resultados de Auto-KGQA e Sequeda et al.

	SQL (SEQUEDA <i>et al.</i> , 2023)	SPARQL (SEQUEDA <i>et al.</i> , 2023)	Auto-QGQA	Improvement	Recall Extracted KG
All Questions	16.7%	54.2%	<b>67.4%</b>	13.2%	91.30%
Low Question/Low Schema	25.5%	71.1%	<b>100%</b>	28.9%	96,87%
High Question/Low Schema	37.4%	66.9%	<b>72.7%</b>	5.8%	91,41%
Low Question/High Schema	0%	35.7%	<b>40%</b>	4.3%	99%
High Question/High Schema	0%	38.5%	<b>50%</b>	11.5%	76,82%

Fonte: elaborada pelo autor.

Ao analisar os resultados é possível observar que Auto-KGQA foi capaz de traduzido corretamente todas as questões da categoria LL. Porém, questões que utilizam muitas tabelas (convertidas em classes no KG) ainda provaram ser um desafio para a Auto-KGQA, tendo em vista o ainda baixa desempenho em questões do tipo LH e HH com acurácia de 40% e 50%. É interessante notar que assim como em (SEQUEDA *et al.*, 2023), as perguntas da categoria LH obtiveram o pior desempenho, ao invés das da categoria HH, como seria esperado. Um fator para consultas da categoria LH terem uma menor acurácia do que as HH é o fato de que estas em geral possuem mais variáveis na projeção do *SELECT*, enquanto questões HH, em geral, possuem funções de agregação como seu resultado. Isto torna mais provável que o LLM esqueça de incluí-las no retorno da consulta.

No geral, as perguntas que tiveram o pior desempenho foram as categorizadas como *High Schema* (LH e HH), com a principal fonte de erros o uso de caminhos incorretos ou incompletos na consulta SPARQL, indicando que o LLM tem dificuldades em manter a consistência em caminhos maiores. Tal desafio é ainda mais claro no cenário utilizando SQL apresentado em (SEQUEDA *et al.*, 2023), onde o LLM errou todas as consultas do tipo *High Schema*.

Tabela 9 – Comparação de tokens do KG usados na entrada

	#Tokens	Média de #Tokens do em Auto-KGQA	Redução média
<b>KG</b>	6,554	3,203.27	51.12%
<b>KG + Inferences</b>	25,460	3,203.27	87.41%

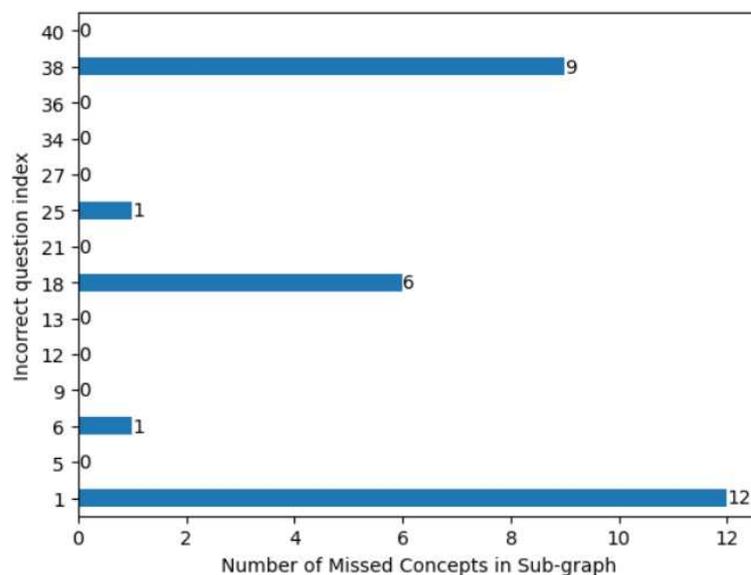
Fonte: elaborada pelo autor.

A tabela 9 compara o número de *tokens* do KG passados ao LLM. Ao analisar o número de *tokens* utilizados por Auto-KGQA também foi possível notar uma considerável

redução em seu número. Em média o *framework* passou 3203 *tokens* do KG, enquanto o KG sem inferências é formado por 6554 e por 25460 ao adicionar as inferências. Deste modo, o *framework* alcançou uma redução de 51.12% na quantidade de *tokens* considerando o KG sem inferências e 87.41% ao adicionar-las. É interessante notar que mesmo embora a taxa máxima de manutenção de triplas foi configurada para 90%, a redução foi muito maior do que 10% por conta do *threshold* de relevância mínima, pois verbalizações que tenham uma relevância a baixo de *min\_relevance\_threshold* não entram no conjunto filtrado, mesmo que ainda haja espaço dentro do número máximo de manutenção.

Para avaliar a taxa de cobertura dos conceitos necessários para escrever a consulta SPARQL retornada pelo algoritmo de extração de sub-grafo relevante (Algoritmo 4) foi desenvolvida uma estratégia automática para calcular seu *recall*. Nesta abordagem, cada consulta SPARQL do *gold-standard* é analisada, onde é extraído todo o conjunto de classes e propriedades nela contidas. Em seguida, cada conceito é pesquisado no sub-grafo retornado, onde é contado o número de conceitos faltantes. O *recall* ( $recall = (C - C')/C$ ) de cada sub-grafo é calculada com base no número total de conceitos esperados,  $C$ , e no número de conceitos esperados presentes nele,  $C'$ . O *recall* médio para todas as questões foi de 91,30%. A Tabela 8 mostra a recordação para cada categoria de questão individualmente. Tais resultados indicam que o algoritmo de extração possui um alto índice de eficiência na seleção dos termos relevantes, ao mesmo tempo em que garante uma eficiência média na redução do sub-grafo gerado.

Figura 22 – Número de conceitos perdidos no sub-grafo para cada uma das perguntas erradas.

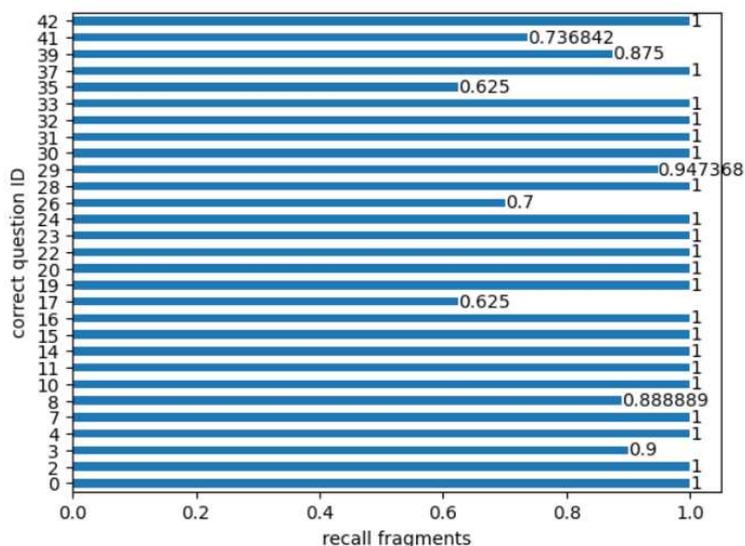


Fonte: Próprio autor.

Ao avaliar os resultados para cada categoria, é curioso notar que a categoria com o maior *recall* (99%) foi a categoria com a pior acurácia (40%), sendo esta a categoria LH. Isto indica que a principal fonte de erros do sistema não foi a passagem de um sub-grafo incompleto para o LLM, mas sim a complexidade inerente da questão. Tendo em vista que as questões do tipo HH obtiveram uma acurácia ligeiramente superior (50%), embora tenham obtido um *recall* muito inferior (76,82%), reforça a ideia de que o processo de filtragem não foi a principal fonte de erros. Isto fica ainda mais evidente ao avaliar o número de conceitos faltantes nos sub-grafos extraídos das questões respondidas incorretamente, onde apenas 35% (5) das questões erradas (14) possuíam um sub-grafo insuficiente. A Figura 22 deixa evidente que apenas um número reduzido de questões respondidas incorretamente possuíam um sub-grafo insuficiente.

Além disso, ao avaliar o *recall* das questões respondidas corretamente (Figura 23) é possível notar que nem todas possuíam todos os recursos esperados no sub-grafo selecionado. Das 29 respostas corretas, 8 possuem um *recall* inferior a 100% (27% do total de corretas), indicando que o LLM pode ser capaz de inferir propriedades ou classes não conhecidas, seguindo o padrão das já conhecidas.

Figura 23 – Recall para perguntas respondidas corretamente.



Fonte: Próprio autor.

As Tabelas 10 e 11 apresentam a análise dos erros presentes nas consultas geradas pelo *framework*.

O Código-fonte 7 apresenta um exemplo de consulta esperada no gold-standard para a questão “What are the total loss, which is the sum of loss payment, loss reserve, expense payment, expense reserve amount by catastrophe and policy number?” respondida incorretamente. Por sua vez, o Código-fonte 8 apresenta o código gerado pelo *framework*.

Código-fonte 7 – Consulta SPARQL do gold-standard para a questão “What are the total loss, which is the sum of loss payment, loss reserve, expense payment, expense reserve amount by catastrophe and policy number?”

```

1 PREFIX : <https://myinsurancecompany.linked.data.world/d/
   chat-with-the-data-benchmark/>
2 PREFIX in: <http://data.world/schema/insurance/>
3 SELECT
4   ?policynumber
5   ?catastropheName
6   (sum(?loss) as ?TotalLoss)
7 WHERE {
8   {
9     ?pcd a in:PolicyCoverageDetail;
10      in:hasPolicy ?policy.
11
12     ?policy a in:Policy;
13      in:policyNumber ?policynumber;
14     .
15
16     ?claim in:against ?pcd.
17      ?claim rdf:type in:Claim;
18      in:claimNumber ?ClaimNumber;
19      in:hasCatastrophe [in:catastropheName ?
20      catastropheName ];
21      in:hasLossPayment [
22      in:lossPaymentAmount ?LossPaymentAmount;
23      in:hasLossReserve [

```

Tabela 10 – Análise dos erros nas consultas do benchmark

Pergunta	Tipo	Sub-graph Recall	Motivo do erro
What are the total loss, which is the sum of loss payment, loss reserve, expense payment, expense reserve amount by catastrophe and policy number?	HH	29,41%	A principal diferença da consulta é a função de agregação no select. Além disso, a consulta gerada é menos restrita.
What are the loss payment, loss reserve, expense payment, expense reserve amount by claim number and corresponding policy number, policy holder, premium amount paid, the catastrophe it had, and the agent who sold it?	LH	100%	Consulta gerada totalmente errada. Apenas uma agregação de um select simples.
How many claims have been placed by policy number?	HL	83,33%	Uma consulta simples porem errou o caminho da consulta. Faltou um item no caminho.
Return policy holders and the claims they have made and the corresponding catastrophe	LH	100%	Erro no caminho. Consulta complexa com necessidade de construção de um caminho longo
What is the average policy size which is the the total amount of premium divided by the number of policies?	HL	100%	Diferença no caminho da consulta.
What is the loss ratio of each policy and agent who sold it by policy number and agent id?	HH	100%	O valor retornado é diferente nas duas consultas.
What are the total loss, which is the sum of loss payment, loss reserve, expense payment, expense reserve amount by claim number, catastrophe and corresponding policy number?	HH	64,47%	Consulta original complexa. Criada com caminho longo e a consulta gerada não consegue montar o caminho.
What is the total loss of each policy that an agent has sold by agent id where total loss is the sum of loss payment, loss reserve, expense payment, expense reserve amounts?	HH	100%	Caminho longo e consulta gerada com agregação. Erro na construção do caminho e inclusão de agregação não necessária.
What is the total loss amounts, which is the sum of loss payment, loss reserve amount by claim number?	HL	83%	Consulta não gerada.
Return agents and the policy they have sold that have had a claim and the corresponding catastrophe it had.	LH	100%	Caminho longo incorreto. Consulta complexa
What are the loss payment, loss reserve, expense payment, expense reserve amount by claim number and corresponding policy number, policy holder and premium amount paid?	LH	100%	Caminho longo incorreto. Consulta complexa

Fonte: elaborada pelo autor.

Tabela 11 – Análise dos erros nas consultas do benchmark parte 2

Pergunta	Tipo	Sub-graph Recall	Motivo do erro
Return agents and the policy they have sold that have had a claim and the corresponding catastrophe it had.	LH	100%	Caminho longo. Consulta complexa
What are the loss payment, loss reserve, expense payment, expense reserve amount by claim number and corresponding policy number, policy holder and premium amount paid?	LH	100%	Caminho longo. Consulta complexa
Return agents and the policy they have sold that have had a claim and the corresponding expense payment amount by agent id, policy number and claim number	LH	100%	Consulta complexa. Errou a construção do caminho e não colocou todos os predicados.
What is the total loss of each policy by policy number where total loss is the sum of loss payment, loss reserve, expense payment, expense reserve amounts?	HH	35,71%	Não acertou o caminho. Consulta complexa e longa e não gerou agregação presente na consulta.
Return agents and the policies they have sold that have had a claim and the corresponding loss payment amount by agent id, policy number and claim number	LH	100%	Caminho longo. Consulta complexa. Incluiu predicado que não existe

Fonte: elaborada pelo autor.

```

24         in:lossReserveAmount ?LossReserveAmount;
25     ];
26     in:hasExpensePayment [
27         in:expensePaymentAmount ?
28         ExpensePaymentAmount;
29     ];
30     in:hasExpenseReserve [
31         in:expenseReserveAmount ?
32         ExpenseReserveAmount;
33     ];
34     .
35     bind (?LossPaymentAmount+ ?LossReserveAmount+ ?
36         ExpensePaymentAmount + ?ExpenseReserveAmount as
37         ?loss)
38 }
39 }
40 group by ?policynumber    ?catastropheName

```

A baixo estão listados os caminhos definidos na consulta do Código-fonte 7:

```

1 in:Claim->in:against->in:PolicyCoverageDetail->in:hasPolicy
   ->in:Policy->in:policyNumber;
2 in:Claim->in:claimNumber;
3 in:Claim->in:hasCatastrophe->in:catastropheName ;
4 in:Claim->in:hasLossPayment->in:lossPaymentAmount;
5 in:Claim->in:hasLossReserve->in:lossReserveAmount;
6 in:Claim->in:hasExpensePayment->in:expensePaymentAmount;
7 in:Claim->in:hasExpenseReserve->in:expenseReserveAmount;

```

Código-fonte 8 – Consulta SPARQL gerada para a questão “What are the total loss, which is the sum of loss payment, loss reserve, expense payment, expense reserve amount by catastrophe and policy number?”

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX ns1: <http://data.world/schema/insurance/>
5
6 SELECT ?catastrophe ?policyNumber (SUM(?lossPaymentAmount +
   ?lossReserveAmount + ?expensePaymentAmount + ?
   expenseReserveAmount) AS ?totalLoss)
7 WHERE {
8   ?catastrophe rdf:type ns1:Catastrophe.
9   ?policy rdf:type ns1:Policy; ns1:policyNumber ?
   policyNumber.
10  ?claim ns1:hasLossPayment ?lossPayment;
11         ns1:hasLossReserve ?lossReserve;
12         ns1:hasExpensePayment ?expensePayment;
13         ns1:hasExpenseReserve ?expenseReserve.
14  ?lossPayment ns1:lossPaymentAmount ?lossPaymentAmount.
15  ?lossReserve ns1:lossReserveAmount ?lossReserveAmount.
16  ?expensePayment ns1:expensePaymentAmount ?
   expensePaymentAmount.
17  ?expenseReserve ns1:expenseReserveAmount ?
   expenseReserveAmount.
18 }
19 GROUP BY ?catastrophe ?policyNumber

```

A baixo estão listados os caminhos definidos na consulta do Código-fonte 8:

```

1 ns1:Catastrophe
2 ns1:Policy->ns1:policyNumber;

```

```

3 in:Claim->ns1:hasLossPayment->ns1:lossPaymentAmount;
4 in:Claim->ns1:hasLossReserve->ns1:lossReserveAmount;
5 in:Claim->ns1:hasExpensePayment->ns1:expensePaymentAmount;
6 in:Claim->ns1:hasExpenseReserve->ns1:expenseReserveAmount;

```

Ao analisar os dois caminhos gerados é possível notar que os caminhos gerados pelo LLM são restritos a caminhos de no máximo tamanho três. Dentre os trechos faltantes destacam-se a ausência do caminho que liga os conceitos *Claim* e *Policy*, sendo este o maior caminho definido na consulta:

```

1 in:Claim->in:against->in:PolicyCoverageDetail->in:hasPolicy
  ->in:Policy->in:policyNumber

```

Além disso, a consulta gerada não define a recuperação da propriedade *claimNumber* e o caminho para recuperar a propriedade *catastropheName*:

```

1 in:Claim->in:claimNumber;
2 in:Claim->in:hasCatastrophe->in:catastropheName;

```

Esta consulta exemplifica um dos principais desafios encontrados ao utilizar o ChatGPT como ferramenta de KGQA, sua aparente “falta” de memória para consultas e caminhos mais longos.

### 6.3 SciQA Dataset

Esta seção apresenta o uso do *dataset* SciQA (AUER *et al.*, 2023) para avaliar o impacto da abordagem *few-shot learning* no desempenho do *framework* Auto-KGQA. O *dataset* conta com um total de 2,565 pares de perguntas e consultas sobre o KG ORKG (JARADEH *et al.*, 2019). O *dataset* é dividido em 70% para treinamento (1795 amostras), 10% para validação (257 amostras) e 20% para teste (513 amostras). Por sua vez, o KG ORKG conta com 2785943 triplas, tendo seu *dump* na serialização n-triples totalizando 373 MB. Deste modo, considerando o grande número de pares de exemplos de consultas de treinamento, além do tamanho do KG subjacente, o SciQA apresenta-se como uma excelente oportunidade para a utilização dos exemplos de treinamento em estratégias de *few-shot learning* em KGs de médio-grande porte.

Por conta de limitações de recursos, nestes experimentos não foram utilizadas

Tabela 12 – Resultados do experimentos no dataset SciQA

Abordagem	F1 (0-shot learning)	F1 (3-shot learning)	Taxa de cobertura do KG necessário
(LEHMANN <i>et al.</i> , 2024) (GPT 3.5 Turbo)	0.2632	0.9667	0%
Auto-KGQA (GPT 3.5 Turbo)	0.7333	0.9	100%

Fonte: elaborada pelo autor.

todas as 513 perguntas do *dataset* de teste. Ao invés disso, foram selecionadas 100 perguntas aleatoriamente, seguindo a mesma proporção de perguntas geradas automaticamente (96%) e manualmente (4%) do *dataset* de teste completo. Além disso, nestes experimentos não foram realizados experimentos exaustivos, concentrando-se em duas configurações de experimentos: (1) Desempenho do *framework* em um abordagem *0-shot learning*<sup>20</sup> e (2) Desempenho do *framework* e uma abordagem *3-shot learning*<sup>21</sup>. Nestes experimentos o modelo de LLM utilizado foi GPT 3.5 Turbo, tendo como parâmetros do *framework* os seguintes valores:

- *edge\_hops*: 5;
- *max\_edges\_by\_property*: 1 (por conta do grande tamanho do KG);
- *min\_relevance\_threshold*: 0.45;
- *maximun\_number\_results*: 10%.

A Tabela 12 apresenta os resultados obtidos pelo Auto-KGQA comparados aos resultados obtidos por (LEHMANN *et al.*, 2024). Para a comparação foi escolhida a abordagem utilizando o GPT 3.5 Turbo, tendo em vista que este é o único modelo cujas duas abordagens possuem suporte em comum. A métrica utilizada para avaliação de desempenho utilizada foi a F1-Score. A última coluna da tabela mede a taxa de cobertura dos recursos do KG necessários para responder a pergunta passados ao LLM como definida na Seção 6.2.

De acordo com os resultados apresentados, é possível notar que embora Auto-KGQA (0.9) não tenha conseguido superar os resultados apresentados por (LEHMANN *et al.*, 2024) (0.9667) no cenário *3-shot learning*, este não foi o caso na configuração *0-shot*. No cenário *0-shot learning*, Auto-KGQA alcançou uma F1 de 0.7333, enquanto (LEHMANN *et al.*, 2024) alcançou apenas 0.2632. Isto se dar por conta do fato de (LEHMANN *et al.*, 2024) depender exclusivamente dos exemplos do conjunto de treinamento para passar conhecimento ao LLM sobre o KG. Vale notar que nos experimentos realizados, todas as consultas geraram resultados completamente ou corretos ou incorretos, desta maneira ocasionando na F1-Score sendo igual

<sup>20</sup> [https://drive.google.com/file/d/1QxCj\\_gdkHtL1sWldNfim3RoEHvHjR1Fn/view?usp=drive\\_link](https://drive.google.com/file/d/1QxCj_gdkHtL1sWldNfim3RoEHvHjR1Fn/view?usp=drive_link)

<sup>21</sup> <https://docs.google.com/spreadsheets/d/1uUANa8-uQGDbOBOAx2iNqX5uKdtKvYcHfx1g3dbnzi4/edit?usp=sharing>

Tabela 13 – F1-Score de Auto-KGQA por tipo de questão

Abordagem	F1 (Manuais)	F1 (Automáticas)
<i>0-shot learning</i>	0.0	0.7674418604651163
<i>3-shot learning</i>	0.5	0.9166666666666666

Fonte: elaborada pelo autor.

à acurácia do modelo. Na abordagem de (LEHMANN *et al.*, 2024) não existe a passagem de fragmentos do KG para o LLM, resultando no valor de 0% apresentado na coluna de taxa de cobertura. Por sua vez, Auto-KGQA seleciona de maneira autônoma o sub-grafo relevante para a questão e passando-o para o LLM, garantido um contexto sobre o KG consultado, mesmo em cenários *0-shot*. Nos testes realizados, o mecanismo de seleção do sub-grafo foi capaz de filtrar triplas que continham 100% dos termos necessários para a consulta SPARQL seguindo a mesma métrica definida na Seção 6.2.

Ao calcular o impacto da passagem dos exemplos nas abordagens, é possível notar que para (LEHMANN *et al.*, 2024), o uso dos exemplos garantiu uma melhora de 0,7035 na F1, mostrando a grande dependência dos exemplos na abordagem. Por sua vez, o uso dos exemplos propiciou uma melhora de 0,1667 na F1 de Auto-KGQA, evidenciando que embora a abordagem não seja tão dependente dos exemplos, estes tem um grande impacto positivo nos resultados.

A Tabela 13 apresenta os resultados dos experimentos de Auto-KGQA por tipo de questão (geradas manualmente ou automaticamente). Nestes resultados é possível notar que na configuração *0-shot* o *framework* não foi capaz de acertar nenhuma questão gerada manualmente, enquanto teve uma F1 de 0.767 para as geradas automaticamente. Isto indica que as do primeiro tipo são muito mais desafiadoras do que as do segundo. Ao avaliar a configuração *3-shot*, é possível notar que o *framework* conseguiu uma F1 de 0.5 para as perguntas geradas manualmente, alcançado 0.9166 para as automáticas. (LEHMANN *et al.*, 2024) não apresenta seus resultados separados por tipo de pergunta, no entanto é possível notar que a F1 apresentada pelo modelo (0.9667) é praticamente o mesmo valor da taxa de questões geradas automaticamente no *dataset* de teste (96% ou 0.96).

Ao analisar os resultados do sistema de seleção de exemplos de perguntas é possível notar que os exemplos existente são praticamente idênticos às perguntas geradas automaticamente, mudando apenas valores de filtros. No entanto, o mesmo não é verdade para as geradas manualmente. A baixo estão os exemplos selecionados para uma questão automática:

- **Questão gerada automaticamente:** *Which model has achieved the highest Accuracy score on the Story Cloze Test benchmark dataset?*

– **SPARQL esperado:**

```

1 SELECT DISTINCT ?model ?model_lbl
2 WHERE {
3   ?metric      a      orkgc:Metric;
4               rdfs:label  ?metric_lbl.
5   FILTER (str(?metric_lbl) = "Accuracy")
6   {
7     SELECT ?model ?model_lbl
8     WHERE {
9       ?dataset      a      orkgc:Dataset;
10                  rdfs:label  ?dataset_lbl.
11      FILTER (str(?dataset_lbl) = "Story Cloze Test")
12      ?benchmark    orkgp:HAS_DATASET      ?dataset;
13                  orkgp:HAS_EVALUATION    ?eval.
14      ?eval          orkgp:HAS_VALUE       ?value;
15                  orkgp:HAS_METRIC        ?metric.
16      ?cont          orkgp:HAS_BENCHMARK   ?benchmark
17                  ;
18                  orkgp:HAS_MODEL         ?model.
19      ?model         rdfs:label           ?model_lbl.
20    }
21    ORDER BY DESC(?value)
22    LIMIT 1
23  }
}

```

– **Exemplos seleccionados:**

```

1 Question: Which model has achieved the highest Accuracy
2           score on the CUB-200-2011 benchmark dataset?
3 Answer: ```sparql
4 PREFIX   orkgc: <http://orkg.org/orkg/class/>
5 PREFIX   orkgp: <http://orkg.org/orkg/predicate/>

```

```

5 PREFIX    orkgr: <http://orkg.org/orkg/resource/>
6
7 SELECT DISTINCT ?model ?model_lbl
8 WHERE {
9   ?metric      a          orkgc:Metric;
10              rdfs:label  ?metric_lbl.
11   FILTER (str(?metric_lbl) = "Accuracy")
12   {
13     SELECT ?model ?model_lbl
14     WHERE {
15       ?dataset      a          orkgc:Dataset;
16                   rdfs:label  ?dataset_lbl.
17       FILTER (str(?dataset_lbl) = "CUB-200-2011")
18       ?benchmark    orkgp:HAS_DATASET      ?dataset;
19                   orkgp:HAS_EVALUATION    ?eval.
20       ?eval          orkgp:HAS_VALUE       ?value;
21                   orkgp:HAS_METRIC       ?metric.
22       ?cont          orkgp:HAS_BENCHMARK   ?benchmark
23                   ;
24                   orkgp:HAS_MODEL        ?model.
25       ?model        rdfs:label            ?model_lbl.
26     }
27     ORDER BY DESC(?value)
28     LIMIT 1
29   }
30 } ````
-----
31 Question: Which model has achieved the highest Accuracy
           score on the OpenBookQA benchmark dataset?
32 Answer: ````sparql
33 PREFIX    orkgc: <http://orkg.org/orkg/class/>
34 PREFIX    orkgp: <http://orkg.org/orkg/predicate/>

```

```

35 PREFIX    orkgr: <http://orkg.org/orkg/resource/>
36
37 SELECT DISTINCT ?model ?model_lbl
38 WHERE {
39     ?metric      a          orkgc:Metric;
40                 rdfs:label  ?metric_lbl.
41     FILTER (str(?metric_lbl) = "Accuracy")
42     {
43         SELECT ?model ?model_lbl
44         WHERE {
45             ?dataset      a          orkgc:Dataset;
46                         rdfs:label  ?dataset_lbl.
47             FILTER (str(?dataset_lbl) = "OpenBookQA")
48             ?benchmark    orkgp:HAS_DATASET      ?dataset;
49                         orkgp:HAS_EVALUATION    ?eval.
50             ?eval         orkgp:HAS_VALUE       ?value;
51                         orkgp:HAS_METRIC        ?metric.
52             ?cont         orkgp:HAS_BENCHMARK   ?benchmark
53                         ;
54                         orkgp:HAS_MODEL         ?model.
55             ?model       rdfs:label            ?model_lbl.
56         }
57         ORDER BY DESC(?value)
58         LIMIT 1
59     }
60 }
-----
61 Question: Which model has achieved the highest Accuracy
        (%) score on the Hendrycks Test benchmark dataset?
62 Answer: ```sparql
63 PREFIX    orkgc: <http://orkg.org/orkg/class/>
64 PREFIX    orkgp: <http://orkg.org/orkg/predicate/>

```

```

65 PREFIX    orkgr: <http://orkg.org/orkg/resource/>
66
67 SELECT DISTINCT ?model ?model_lbl
68 WHERE {
69     ?metric      a          orkgc:Metric;
70                 rdfs:label  ?metric_lbl.
71     FILTER (str(?metric_lbl) = "Accuracy (%)")
72     {
73         SELECT ?model ?model_lbl
74         WHERE {
75             ?dataset      a          orkgc:Dataset;
76                         rdfs:label  ?dataset_lbl.
77             FILTER (str(?dataset_lbl) = "Hendrycks Test")
78             ?benchmark    orkgp:HAS_DATASET      ?dataset;
79                         orkgp:HAS_EVALUATION    ?eval.
80             ?eval         orkgp:HAS_VALUE        ?value;
81                         orkgp:HAS_METRIC        ?metric.
82             ?cont         orkgp:HAS_BENCHMARK    ?benchmark
83                         ;
84                         orkgp:HAS_MODEL         ?model.
85             ?model        rdfs:label            ?model_lbl.
86         }
87         ORDER BY DESC(?value)
88         LIMIT 1
89     }
90 }

```

Neste caso é possível notar que a primeira consulta de exemplo retornada já é exatamente igual a consulta desejada, mudando apenas o valor do filtro `FILTER (str(?dataset_lbl) = "Story Cloze Test")` por `FILTER (str(?dataset_lbl) = "CUB-200-2011")`. O mesmo padrão se repete para as demais consultas geradas automaticamente, indicando que caso o LLM simplesmente faça pequenas alterações nos exemplos passados, ele será capaz de aceitar todas as

consultas deste tipo. Considerando isto, mais o fato de que o F1 de (LEHMANN *et al.*, 2024) foi praticamente o mesmo número de questões deste tipo, é provável que o LLM apenas acertou as questões geradas automaticamente. A baixo segue a consulta final gerada pelo *framework* Auto-KGQA para o caso apresentado anteriormente, resultando numa consulta correta para a abordagem:

```

1 SELECT DISTINCT ?model ?model_lbl
2 WHERE {
3   ?metric      a      orkgc:Resource;
4               rdfs:label ?metric_lbl.
5   FILTER (REGEX(?metric_lbl,"Accuracy","i"))
6   {
7     SELECT ?model ?model_lbl
8     WHERE {
9       ?dataset      a      orkgc:Dataset;
10                  rdfs:label      ?dataset_lbl.
11      FILTER (REGEX(?dataset_lbl,"Story Cloze Test","i"))
12      ?benchmark      ns1:HAS_DATASET      ?dataset;
13                      ns1:HAS_EVALUATION  ?eval.
14      ?eval            ns1:HAS_VALUE      ?value;
15                      ns1:HAS_METRIC      ?metric.
16      ?cont            ns1:HAS_BENCHMARK  ?benchmark;
17                      ns1:HAS_MODEL      ?model.
18      ?model          rdfs:label      ?model_lbl.
19    }
20   ORDER BY DESC(?value)
21   LIMIT 1

```

A baixo seguem os exemplos selecionados para uma questão gerada manualmente:

- **Questão gerada manualmente:** *What is the most common Knowledge representation method?*
- **SPARQL esperado:**

```

1 SELECT (COUNT(?repr) AS ?cnt) ?repr

```

```

2 WHERE {
3   orkgr:R8364 orkgp:compareContribution ?cont.
4   ?cont orkgp:P7046 ?sys.
5   ?sys orkgp:P7056 ?repr.
6 }
7 GROUP BY ?repr
8 ORDER BY DESC(?cnt)
9 LIMIT 1

```

– **Exemplos seleccionados:**

```

1 question: Which are five the most common research
   fields for papers?
2 SPARQL:```sparql
3 PREFIX   orkgc: <http://orkg.org/orkg/class/>
4 PREFIX   orkgp: <http://orkg.org/orkg/predicate/>
5 PREFIX   orkgr: <http://orkg.org/orkg/resource/>
6
7 SELECT ?research_field, ?research_field_labels
8 WHERE {
9   ?papers orkgp:P30 ?research_field.
10  ?research_field rdfs:label ?research_field_labels.
11 }
12 ORDER BY DESC(COUNT(?research_field_labels))
13 LIMIT 5 ```
14 -----
15 question: What is the most common drug in the studies?
16 SPARQL:```sparql
17 PREFIX   orkgc: <http://orkg.org/orkg/class/>
18 PREFIX   orkgp: <http://orkg.org/orkg/predicate/>
19 PREFIX   orkgr: <http://orkg.org/orkg/resource/>
20
21 SELECT ?drug, ?drug_labels

```

```

22 WHERE {
23   orkgr:R155621 orkgp:compareContribution ?contrib.
24   ?contrib orkgp:P37578 ?drug.
25   ?drug rdfs:label ?drug_labels.
26 }
27 ORDER BY DESC(COUNT(?drug))
28 LIMIT 1 ````
29 -----
30 question: What is the most common substrate for
31   catalysts?
32 SPARQL:````sparql
33 PREFIX   orkgc: <http://orkg.org/orkg/class/>
34 PREFIX   orkgp: <http://orkg.org/orkg/predicate/>
35 PREFIX   orkgr: <http://orkg.org/orkg/resource/>
36
37 SELECT ?substrate
38 WHERE {
39   orkgr:R25900 orkgp:compareContribution ?contrib.
40   ?contrib orkgp:P15090 ?substrate.
41 }
42 ORDER BY DESC(COUNT(?substrate))
43 LIMIT 1 ````

```

Neste caso é possível notar que embora os enunciados das questões sejam relativamente parecidos, o conteúdo das consultas SPARQL não é tão parecido como ocorre no caso das automáticas. Deste modo, este tipo de questão torna-se muito mais desafiadora para o LLM. A baixo segue a consulta final gerada pelo *framework* Auto-KGQA para o caso apresentado anteriormente, resultando numa consulta incorreta para a abordagem:

```

1 SELECT ?representation_method ?label
2 WHERE {
3   ?resource orkgp:P7056 ?representation_method.
4   ?representation_method rdfs:label ?label.

```

```

5 }
6 ORDER BY DESC(COUNT(?representation_method))
7 LIMIT 1

```

A quantidade de *tokens* média dos sub-grafos selecionados do KG é de 10152. Por conta do grande tamanho de ORKG, torna-se inviável calcular a quantidade de *tokens* necessários para representar todo o KG. Além disso, também não é possível ter números exatos para representar apenas o T-Box, tendo em vista que o *dump* disponível do ORKG não faz a distinção explícita entre T-Box e A-Box. No entanto, uma estimativa grosseira do número de *tokens* existentes no T-Box de ORKG é de aproximadamente 452393 *tokens*. Esta estimativa é feita com base na seleção apenas das triplas em que as 2008 Classes e 10526 Propriedades de ORKG aparecem como sujeito ou predicados de triplas. Deste modo, a redução proporcionada pelo mecanismo de filtragem de Auto-KGQA foi de no mínimo 97,75%, tendo em vista que a quantidade média de *tokens* selecionados representa apenas 2,24% do tamanho de uma estimativa grosseira do T-Box do KG.

#### 6.4 Discussões sobre os Resultados

Ao comparar os resultados obtidos nas Seções 6.1 e 6.2 é possível notar uma clara diferença no desempenho do *framewrok*, onde no primeiro caso o sistema obteve uma acurácia de 93%, enquanto no segundo alcançando uma acurácia de apenas 67.4%. Embora tal discrepância possa a primeira vista colocar em cheque a consistência e confiabilidade no desempenho da abordagem adotada, ao analisar a natureza das questões adotadas nos dois *benchmarks* é torna-se visível que na realidade os sistema comportou-se de forma consistente. Isto ocorre porque ao tentar classificar as questões apresentadas no primeiro *benchmark* de acordo com a classificação do segundo, torna-se claro que estas seriam classificadas como questões do tipo *Low Schema* (LL e HL). Questões deste tipo utilizam de 0 até 4 tabelas, podendo estas serem interpretadas como caminhos de até tamanho 4 numa consulta SPARQL, o que encaixa-se com as consultas definidas no primeiro conjunto de questões. Tendo isto em vista, ao comparar o desempenho do *framework* com relação as consultas do primeiro *benchmark*, com acurácia de 93%, e as questões do tipo *Low Schema* do segundo, com acurácia de 87% (20 questões respondidas corretamente das 23), nota-se um desempenho muito semelhante do sistema.

As consultas contidas no *dataset* SciQA utilizadas nos experimentos da Seção

6.3 também são classificadas como do tipo *Low Schema*, no entanto, o vocabulário seguido pelo KG ORKG possui o agravante de usar URIs codificadas. Uma URI codificada trata-se de uma URI cuja sua semântica não é explícita para humanos, utilizando códigos ao invés de nomes de conceitos, por exemplo, as URIs `<http://orkg.org/orkg/predicate/P149068>` e `<http://orkg.org/orkg/predicate/P149065>` representam respectivamente as propriedades “*wind rose presence*” e “*Session length*”. Este pode ser um dos motivos para o LLM alcançar uma menor acurácia nos experimentos em *0-shot learning*, obtendo uma acurácia de 73.33%.

No geral, LLMs parecem ter uma maior dificuldade em escrever consultas sobre esquemas muito largos, isto é, esquemas que possuam muitas tabelas ou conceitos em que precisem ser aplicadas junções. Isto pode indicar que modelos mais compactos, onde os dados comumente mais consultados encontram-se mais próximos, sejam mais indicados ao se trabalhar com LLMs. Em (NASCIMENTO *et al.*, 2024a; NASCIMENTO *et al.*, 2024b), Nascimento et al. reforça esta ideia, onde foram geradas visões SQL como uma única tabela com todos os dados necessários para responder a consulta, o que ajudou a contornar as dificuldades do LLM de gerar consultas com *JOINS* corretos.

Uma possível solução para este desafio que não envolva a alteração do esquema a ser consultado pode ser a passagem de exemplos destes caminhos ou *JOINS* mais longos para o LLM via prompt. Esta passagem pode ser feita utilizando técnicas de *Retrieval-Augmented Generation* (RAG) (LEWIS *et al.*, 2020) ou *few-shot learning* em cenários onde existe um conjunto de exemplos bem curados ou através da pre-computação de possíveis caminhos ou *JOINS* envolvendo os conceitos ou tabelas citados na consulta. Esta segunda abordagem pode ser alcançada através da passagem da *steiner-tree* envolvendo os nós dos termos da questão, embora esta possa não representar especificamente o caminho semanticamente desejado na pergunta do usuário.

Embora possíveis soluções possam ser elaboradas para tratar consultas que utilizem grandes quantidades de tabelas ou conceitos, vale a pena ponderar sobre a validade deste tipo de consultas em cenários de mundo real. Interfaces de linguagem natural são costumeiramente utilizadas por usuários leigos ou não técnicos para realizar atividades pontuais (costumeiramente simples) do dia-a-dia. Com isto em vista, talvez consultas tão complexas não se encaixem no tipo de carga de trabalho para qual tais sistemas de KGQA são pretendidos. Conseqüentemente, é necessário que haja diretrizes mais claras e definidas sobre o tipo de questões desejáveis para serem utilizadas em futuros *benchmarks* para sistemas de KGQA.

Além disso, é possível notar que a “criatividade” do LLM representa tanto um risco como uma oportunidade no contexto de KGQA. Em contextos de QA inteiramente em linguagem natural, respostas mais “criativas” podem ser completamente erradas e indistinguíveis de respostas corretas. No entanto, no contexto de KGQA, a criatividade pode ajudar ao LLM a inferir uma propriedade ou classe ausente no grafo de referência passado, mas que pode ser computada durante a execução da consulta. Por sua vez, erros seriam facilmente reconhecidos, tendo em vista o provável não funcionamento da consulta gerada. Mesmo em casos em que o LLM gere uma consulta válida, mas que não represente fielmente a intenção do usuário, a validade da resposta pode ser auferida ao analisar a consulta gerada. Deste modo, uma das vantagens da geração de consultas intermediárias é explicabilidade dos resultados.

Por fim, os experimentos apresentados na Seção 6.3 mostram o impacto positivo da passagem de exemplos de consultas para o LLM, garantindo uma melhora expressiva no desempenho destas na tarefa de KGQA. Isto ocorre mesmo quando considerando a já existência da passagem de fragmentos do KG via *prompt*, como no caso do *framework* Auto-KGQA. No entanto, no caso do *dataset* SciQA, onde as consultas do conjunto de treinamento são apenas variações das do conjunto de teste, é visível que o LLM obteve melhores resultados quando recendo apenas estas exemplos, tendo em vista os resultados de (LEHMANN *et al.*, 2024). Isto pode ocorrer por o sub-grafo passado tirar o foco do LLM dos exemplos passados, introduzindo classes e propriedades que são irrelevantes para a consulta. No entanto, como apresentado na mesma seção, recorrer apenas ao uso de estratégias de *few-shot learning* traria um pobre desempenho em cenários onde não existem tais exemplos, tais como os apresentados nas Seções 6.1 e 6.2. Deste modo, tendo como objetivo sistemas mais generalizáveis, o uso da passagem de fragmentos do KG, acompanhados de exemplos de consultas demonstra-se como uma abordagem promissora.

Tendo em vista todas as observações apresentadas nesta tese, a baixo segue as respostas para as Questões de Pesquisa Gerais (QPGs) definidas nesta tese:

- *QPG*<sub>1</sub>: Como responder questões em linguagem natural sobre um KG utilizando LLMs?
- Resposta: A estratégia mais popular para utilizar LLMs como ferramentas de KGQA encontrada na comunidade e adotada nesta pesquisa foi a passagem do KG sendo consultado como entrada via *prompt* para o LLM, gerando uma interpretação intermediária para uma linguagem formal de consulta, tal como SPARQL. Neste *prompt* o KG é passado diretamente em um formato de serialização padrão, tal como n-triples ou RDF Turtle. Tal

abordagem permite que qualquer KG possa ser consultado sem a necessidade *fine-tuning* ou técnicas mais rebuscadas de ingestão do KG. A geração de consultas em linguagem formal, tal como SPARQL, evita alucinações do LLM com relação a inferência lógicas e computação de funções de agregação, além de permitir a explicabilidade do raciocínio utilizado para gerar o resultado final.

- *QPG<sub>2</sub>*: Como filtrar triplas relevantes para uma questão de um KG?
- Resposta: Na abordagem adotada neste trabalho são selecionadas as triplas que definem recursos e conceitos explicitamente referenciados na questão de entrada. Além disso, as todas as triplas relacionadas com os elementos explicitamente citados e as triplas de seus vizinhos até a profundidade  $N$  (hiper-parâmetro) são avaliadas a serem incluídas. A avaliação destas triplas é feita com base em suas verbalizações, escolhendo as triplas cujas verbalizações possuam um vetor de *embeddings* mais próximo ao vetor de *embeddings* da questão. Por fim, também são adicionadas a triplas intermediárias necessárias para conectar o conjunto de triplas selecionadas.
- *QPG<sub>3</sub>*: Quais os principais desafios para LLMs no cenário de KGQA?
- Resposta: Dentre os principais desafios destacam-se:
  - Como lidar com consultas complexas que descrevam longos caminhos de propriedades.
  - Como garantir que o sub-grafo gerado seja de fato o menor que represente a questão do usuário, tendo em vista que podem existir múltiplos caminhos semanticamente diferentes que conectam um mesmo conjunto de nós. Isto sendo ainda mais agravado ao considerar que o caminho semanticamente correto não é necessariamente o menor caminho a conectá-los.
  - Como tratar a ambiguidade inerente de consultas em linguagem natural.
  - Como identificar alucinações gerados pelo LLM de maneira automática, evitando o retorno de informações falsas ao usuário. Embora o usuário possa analisar a corretude da consulta gerada pelo LLM, esta opção pode demonstra-se inviável para usuários leigos.

A Tabela 14 apresenta as publicações relacionadas ao tema principal desta tese produzidas durante o tempo de pesquisa.

Tabela 14 – Publicações relacionadas ao tema principal de pesquisa

#	Nome do artigo	Local de Publicação	Status	Qualis
1	MediBot: An Ontology-Based Chatbot to Retrieve Drug Information and Compare its Prices	JIDM (Journal)	Publicado	B1
2	LiRB: Um Navegador Leve Baseado em Texto para Knowledge Graphs RDF	SBBB 2023	Publicado	A4
3	<b>Experiments with text-to-SPARQL based on ChatGPT</b>	ICSC 2024	Publicado	A3
4	<b>A Framework for Question Answering on Knowledge Graphs Using Large Language Models</b>	ESWC 2024	Publicado	A2
5	A survey on Interactive Question Answering Systems and Its Interactive Aspects	KIS (Journal)	Em Avaliação	A2
6	<b>An Autonomous Domain-independent Framework Based on LLMs for Text-to-SPARQL</b>	IJSC (Journal)	Aceito para publicação	A2
7	Small, Medium, and Large Language Models for Text-to-SQL: Which is your size?	ER 2024	Aceito para publicação	A3

Fonte: elaborada pelo autor.

Por fim, a Tabela 15 apresenta as publicações secundárias produzidas durante o tempo de pesquisa.

Tabela 15 – Publicações secundárias

#	Nome do artigo	Local de Publicação	Status	Qualis
1	Um Enfoque Incremental para Construção do Grafo de Conhecimento do SUS	SBCAS 2020	Publicado	A4
2	Ontology-based Question Answering Systems over Knowledge Bases: A Survey	ICEIS 2020	Publicado	A3
3	Uso das tecnologias da web semântica na construção de grafos de conhecimento semântico baseado no enfoque híbrido	ONTOBRAS 2021	Publicado	B3
4	KG-E: Um Grafo de Conhecimento Semântico Baseado na Integração de Dados de Empresas e Sancionados	WCGE 2021	Publicado	B4
5	UM ENFOQUE INCREMENTAL PARA CONSTRUÇÃO DO GRAFO DE CONHECIMENTO DO SUS	Capítulo de Livro 2021	Publicado	-
6	DIKW4IoT: Uma abordagem baseada na hierarquia DIKW para a construção de grafos de conhecimento para integração de dados de IoT	SBBB 2022	Publicado	A4
7	ExQuestions: An Expanded Factual Corpus for Question Answering over Knowledge Graphs	ICSC 2022	Publicado	A3
8	Construção do Grafo de Conhecimento Semântico de Dados Abertos de Pessoas Jurídicas	ONTOBRAS 2022	Publicado	B3
9	An Approach to Construct a Metadata Graph for Specifying Enterprise Knowledge Graphs	SBBB 2024	Aceito para publicação	A4
10	Construção do Dataset Semântico de Pessoas Jurídicas	DSW (SBBB) 2024	Aceito para publicação	A4

Fonte: elaborada pelo autor.

## 7 CONCLUSÃO

Nesta tese foi apresentado Auto-KGQA, um *framework* autônomo independente de domínio baseado em LLMs para KGQA. Auto-KGQA tem como seu principal diferencial sua capacidade de selecionar autonomamente sub-grafos do KG relevantes para a questão, fornecendo-os para o LLM como contexto para a geração de consultas. Experimentos aplicados ao *framework* concluíram que a abordagem alcança uma acurácia similar ao passar todo o KG, no entanto, com uma redução de 81.98% no número de *tokens* em comparação com o KG sem inferências e com uma redução de 95.23% no número de *tokens* em comparação com o KG com inferências.

Além disso, experimentos foram realizados com um *benchmark* disponibilizado publicamente para comparar o desempenho de Auto-KGQA contra demais abordagens da comunidade em questões de diferentes níveis de complexidade. Nestes experimentos o *framework* conseguiu melhorias de acurácia de cerca de 13.2% em relação ao melhor competidor, alcançando melhorias em todos os níveis de dificuldade. Além disso, o *framework* conseguiu reduzir em cerca de 51.12% o número de *tokens* do KG passados na entrada, chegando a uma redução de 87.41% quando considerando também o cenário do KG carregado com inferências.

Experimentos revelaram que o uso de estratégias de *few-shot learning* em conjunto com o sub-grafo selecionado por Auto-KGQA geram sistemas de KGQA robustos e generalizáveis capazes de responder, tanto questões já conhecidas, quanto inéditas. Tal abordagem garantiu ao *framework* um desempenho superior ao de sistemas comparáveis em abordagens *0-shot learning*, garantindo uma F1-Score de 0.7333, contra 0.2632 do melhor competidor comparável. Embora o sistema não tenha superado o principal competidor, alcançando 0.9 de F1-Score, contra 0.9667, o sistema ainda assim demonstra-se como uma opção viável em cenários deste tipo.

Com base nestes experimentos foi possível notar que, no geral, LLMs parecem ter uma maior dificuldade em escrever consultas sobre esquemas muito largos, isto é, esquemas que possuam muitas tabelas ou conceitos em que precisem ser aplicadas junções. Isto pode indicar que modelos mais compactos, onde os dados comumente mais consultados encontram-se mais próximos, sejam mais indicados ao se trabalhar com LLMs.

Por fim, nesta tese foi possível observar que LLMs são promissores no campo de KGQA, conseguindo responder uma ampla gama de questões para KGs específicos sem a necessidade de grandes volumes de dados de treinamento. Trabalhos futuros incluem a

implantação do *framework* em cenários de mundo real, além de experimentos utilizando LLMs de código aberto para substituir o ChatGPT da OpenAI.

## REFERÊNCIAS

- ALLAM, A. M. N.; HAGGAG, M. H. The question answering systems: A survey. **International Journal of Research and Reviews in Information Sciences (IJRRIS)**, v. 2, n. 3, 2012.
- ARNDT JEEN BROEKSTRA, B. D. D. **RDF-star and SPARQL-star**. 2022. [https://w3c.github.io/rdf-star/cg-spec/editors\\_draft.html](https://w3c.github.io/rdf-star/cg-spec/editors_draft.html). Accessed: 2023-05-25.
- ATHIWARATKUN, B.; WILSON, A. G.; ANANDKUMAR, A. Probabilistic fasttext for multi-sense word embeddings. **arXiv preprint arXiv:1806.02901**, 2018.
- AUER, S.; BARONE, D. A.; BARTZ, C.; CORTES, E. G.; JARADEH, M. Y.; KARRAS, O.; KOUBARAKIS, M.; MOUROMTSEV, D.; PLIUKHIN, D.; RADYUSH, D. *et al.* The sciqa scientific question answering benchmark for scholarly knowledge. **Scientific Reports**, Nature Publishing Group UK London, v. 13, n. 1, p. 7240, 2023.
- AYOOLA, T.; TYAGI, S.; FISHER, J.; CHRISTODOULOPOULOS, C.; PIERLEONI, A. Refined: An efficient zero-shot-capable approach to end-to-end entity linking. **arXiv preprint arXiv:2207.04108**, 2022.
- BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific american**, JSTOR, v. 284, n. 5, p. 34–43, 2001.
- BIKAKIS, N.; SELLIS, T. Exploration and visualization in the web of big linked data: A survey of the state of the art. **arXiv preprint arXiv:1601.08059**, 2016.
- BLINOV, P. Semantic triples verbalization with generative pre-training model. In: **Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)**. [S. l.: s. n.], 2020. p. 154–158.
- BOLLACKER, K.; EVANS, C.; PARITOSH, P.; STURGE, T.; TAYLOR, J. 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.
- BROWN, T.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J. D.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A. *et al.* Language models are few-shot learners. **Advances in neural information processing systems**, v. 33, p. 1877–1901, 2020.
- CER, D.; YANG, Y.; KONG, S.-y.; HUA, N.; LIMTIACO, N.; JOHN, R. S.; CONSTANT, N.; GUAJARDO-CESPEDES, M.; YUAN, S.; TAR, C. *et al.* Universal sentence encoder. **arXiv preprint arXiv:1803.11175**, 2018.
- CHANG, Y.; WANG, X.; WANG, J.; WU, Y.; ZHU, K.; CHEN, H.; YANG, L.; YI, X.; WANG, C.; WANG, Y. *et al.* A survey on evaluation of large language models. **arXiv preprint arXiv:2307.03109**, 2023.
- DETTMERS, T.; PAGNONI, A.; HOLTZMAN, A.; ZETTLEMOYER, L. Qlora: Efficient finetuning of quantized llms. **arXiv preprint arXiv:2305.14314**, 2023.
- DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.

DIEFENBACH, D.; LOPEZ, V.; SINGH, K.; MARET, P. Core techniques of question answering systems over knowledge bases: a survey. **Knowledge and Information systems**, Springer, v. 55, n. 3, p. 529–569, 2018.

FENG, C.; ZHANG, X.; FEI, Z. Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs. **arXiv preprint arXiv:2309.03118**, 2023.

FIGUEIREDO, J. C. A. de. **Tutorial de Manipulação de Strings**. 2007. <http://www.dsc.ufcg.edu.br/~abrant/CursosAnteriores/ATAL031/String/String.html>. Acesso em 23 de novembro de 2023.

GIUNTI, M.; SERGIOLI, G.; VIVANET, G.; PINNA, S. Representing n-ary relations in the semantic web. **Logic Journal of the IGPL**, Oxford University Press, v. 29, n. 4, p. 697–717, 2021.

GOLDBERG, Y.; LEVY, O. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. **arXiv preprint arXiv:1402.3722**, 2014.

Google. **Google Gemini**. 2024. Acesso em: 21 jun. 2024. Disponível em: <https://gemini.google.com/>.

GOYAL, P.; FERRARA, E. Graph embedding techniques, applications, and performance: A survey. **Knowledge-Based Systems**, Elsevier, v. 151, p. 78–94, 2018.

GROVER, A.; LESKOVEC, J. node2vec: Scalable feature learning for networks. In: **Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining**. [S. l.: s. n.], 2016. p. 855–864.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowledge acquisition**, Elsevier, v. 5, n. 2, p. 199–220, 1993.

HAKIMOV, S.; UNGER, C.; WALTER, S.; CIMIANO, P. Applying semantic parsing to question answering over linked data: Addressing the lexical gap. In: SPRINGER. **International Conference on Applications of Natural Language to Information Systems**. [S. l.], 2015. p. 103–109.

HAMILTON, W.; YING, Z.; LESKOVEC, J. Inductive representation learning on large graphs. **Advances in neural information processing systems**, v. 30, 2017.

HIRSCHMAN, L.; GAIZAUSKAS, R. Natural language question answering: the view from here. **natural language engineering**, Cambridge University Press, v. 7, n. 4, p. 275–300, 2001.

HÖFFNER, K.; WALTER, S.; MARX, E.; USBECK, R.; LEHMANN, J.; NGOMO, A.-C. N. Survey on challenges of question answering in the semantic web. **Semantic Web**, IOS Press, v. 8, n. 6, p. 895–920, 2017.

HU, E. J.; SHEN, Y.; WALLIS, P.; ALLEN-ZHU, Z.; LI, Y.; WANG, S.; WANG, L.; CHEN, W. Lora: Low-rank adaptation of large language models. **arXiv preprint arXiv:2106.09685**, 2021.

HWANG, F. K.; RICHARDS, D. S. Steiner tree problems. **Networks**, Wiley Online Library, v. 22, n. 1, p. 55–89, 1992.

JARADEH, M. Y.; OELEN, A.; FARFAR, K. E.; PRINZ, M.; D'SOUZA, J.; KISMIHÓK, G.; STOCKER, M.; AUER, S. Open research knowledge graph: next generation infrastructure for semantic scholarly knowledge. In: **Proceedings of the 10th international conference on knowledge capture**. [S. l.: s. n.], 2019. p. 243–246.

JIN, D.; PAN, E.; OUFATTOLE, N.; WENG, W.-H.; FANG, H.; SZOLOVITS, P. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. **Applied Sciences**, MDPI, v. 11, n. 14, p. 6421, 2021.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. **arXiv preprint arXiv:1609.02907**, 2016.

KIROS, R.; ZHU, Y.; SALAKHUTDINOV, R. R.; ZEMEL, R.; URTASUN, R.; TORRALBA, A.; FIDLER, S. Skip-thought vectors. **Advances in neural information processing systems**, v. 28, 2015.

KOVRIGUINA, L.; TEUCHER, R.; RADYUSH, D.; MOUROMTSEV, D. Sparqlgen: One-shot prompt-based approach for sparql query generation. 2023.

LEARN, S. **F1 sklearn.metrics.f1\_score**. 2019. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html). Accessed: 2020-01-06.

LEHMANN, J.; GATTOGI, P.; BHANDIWAD, D.; FERRÉ, S.; VAHDAT, S. Language models as controlled natural language semantic parsers for knowledge graph question answering. 2023.

LEHMANN, J.; MELONI, A.; MOTTA, E.; OSBORNE, F.; RECUPERO, D. R.; SALATINO, A. A.; VAHDATI, S. Large language models for scientific question answering: An extensive analysis of the sciq benchmark. In: SPRINGER. **European Semantic Web Conference**. [S. l.], 2024. p. 199–217.

LEWIS, P. *et al.* Retrieval-augmented generation for knowledge-intensive nlp tasks. In: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M.; LIN, H. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2020. v. 33, p. 9459–9474. Disponível em: <https://api.semanticscholar.org/CorpusID:218869575>.

LUO, H.; TANG, Z.; PENG, S.; GUO, Y.; ZHANG, W.; MA, C.; DONG, G.; SONG, M.; LIN, W. *et al.* Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. **arXiv preprint arXiv:2310.08975**, 2023.

MICROSOFT. **Satori Understand Your World with Bing**. 2019. <https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/>. Accessed: 2020-01-03.

MILLER, G. A. Wordnet: a lexical database for english. **Communications of the ACM**, ACM New York, NY, USA, v. 38, n. 11, p. 39–41, 1995.

NASCIMENTO, E. R.; GARCIA, G. M.; FEIJÓ, L.; VICTORIO, W. Z.; LEMOS, M.; IZQUIERDO, Y. T.; GARCIA, R. L.; LEME, L. A. P.; CASANOVA, M. A. Text-to-sql meets the real-world. In: **Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1**. [S. l.: s. n.], 2024. p. 61–72.

NASCIMENTO, E. R.; IZQUIERDO, Y. T.; GARCIA, G. M.; COELHO, G.; FEIJÓ, L.; LEMOS, M.; LEME, L. A. P.; CASANOVA, M. A. My database user is a large language model. In: **Proceedings of the 26th International Conference on Enterprise Information Systems – Volume 1**. [S. l.: s. n.], 2024. p. 800–806.

NGOMO, N. 9th challenge on question answering over linked data (qald-9). **language**, v. 7, n. 1, p. 58–64, 2018.

OUYANG, L.; WU, J.; JIANG, X.; ALMEIDA, D.; WAINWRIGHT, C.; MISHKIN, P.; ZHANG, C.; AGARWAL, S.; SLAMA, K.; RAY, A. *et al.* Training language models to follow instructions with human feedback. **Advances in Neural Information Processing Systems**, v. 35, p. 27730–27744, 2022.

PAN, J. Z.; VETERE, G.; GOMEZ-PEREZ, J. M.; WU, H. **Exploiting linked data and knowledge graphs in large organisations**. [S. l.]: Springer, 2017.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: **Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)**. [S. l.: s. n.], 2014. p. 1532–1543.

PEROZZI, B.; AL-RFOU, R.; SKIENA, S. Deepwalk: Online learning of social representations. In: **Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S. l.: s. n.], 2014. p. 701–710.

QI, Z.; YU, Y.; TU, M.; TAN, J.; HUANG, Y. Foodgpt: A large language model in food testing domain with incremental pre-training and knowledge graph prompt. **arXiv preprint arXiv:2308.10173**, 2023.

QUAN, D.; KARGER, R. How to make a semantic web browser. In: **Proceedings of the 13th international conference on World Wide Web**. [S. l.: s. n.], 2004. p. 255–265.

RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; SUTSKEVER, I. *et al.* Language models are unsupervised multitask learners. **OpenAI blog**, v. 1, n. 8, p. 9, 2019.

RODRIGUEZ-MURO, M.; KONTCHAKOV, R.; ZAKHARYASCHEV, M. Ontology-based data access: Ontop of databases. In: SPRINGER. **The Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I 12**. [S. l.], 2013. p. 558–573.

RONY, M. R. A. H.; KUMAR, U.; TEUCHER, R.; KOVRIGUINA, L.; LEHMANN, J. Sgpt: a generative approach for sparql query generation from natural language questions. **IEEE Access**, IEEE, v. 10, p. 70712–70723, 2022.

SAHA, D.; FLORATOU, A.; SANKARANARAYANAN, K.; MINHAS, U. F.; MITTAL, A. R.; ÖZCAN, F. Athena: an ontology-driven system for natural language querying over relational data stores. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 12, p. 1209–1220, 2016.

SEN, P.; AJI, A. F.; SAFFARI, A. Mintaka: A complex, natural, and multilingual dataset for end-to-end question answering. **arXiv preprint arXiv:2210.01613**, 2022.

SEQUEDA, J.; ALLEMANG, D. T.; JACOB, B. A benchmark to understand the role of knowledge graphs on large language model’s accuracy for question answering on enterprise sql databases. **ArXiv**, abs/2311.07509, 2023. Disponível em: <https://api.semanticscholar.org/CorpusID:265150432>.

SILVA, J. W. F. da; VENCESLAU, A. D. P.; SALES, J. E.; MAIA, J. G. R.; PINHEIRO, V. C. M.; VIDAL, V. M. P. A short survey on end-to-end simple question answering systems. **Artificial Intelligence Review**, Springer, v. 53, n. 7, p. 5429–5453, 2020.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: **Advances in neural information processing systems**. [S. l.: s. n.], 2014. p. 3104–3112.

TAN, Y.; MIN, D.; LI, Y.; LI, W.; HU, N.; CHEN, Y.; QI, G. Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family. In: SPRINGER. **International Semantic Web Conference**. [S. l.], 2023. p. 348–367.

TIAN, Y.; SONG, H.; WANG, Z.; WANG, H.; HU, Z.; WANG, F.; CHAWLA, N. V.; XU, P. Graph neural prompting with large language models. **arXiv preprint arXiv:2309.15427**, 2023.

TOUVRON, H.; LAVRIL, T.; IZACARD, G.; MARTINET, X.; LACHAUX, M.-A.; LACROIX, T.; ROZIÈRE, B.; GOYAL, N.; HAMBRO, E.; AZHAR, F. *et al.* Llama: Open and efficient foundation language models. **arXiv preprint arXiv:2302.13971**, 2023.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

VELICKOVIC, P.; CUCURULL, G.; CASANOVA, A.; ROMERO, A.; LIO, P.; BENGIO, Y. *et al.* Graph attention networks. **stat**, v. 1050, n. 20, p. 10–48550, 2017.

W3. **OWL OWL Web Ontology Language**. 2013. <https://www.w3.org/OWL/>. Accessed: 2019-01-03.

WANG, Q.; MAO, Z.; WANG, B.; GUO, L. Knowledge graph embedding: A survey of approaches and applications. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 29, n. 12, p. 2724–2743, 2017.

WIKIPEDIA. **Prompt Engineering**. 2023. [https://en.wikipedia.org/wiki/Prompt\\_engineering#Text-to-text](https://en.wikipedia.org/wiki/Prompt_engineering#Text-to-text). Recuperado de [https://en.wikipedia.org/wiki/Prompt\\_engineering#Text-to-text](https://en.wikipedia.org/wiki/Prompt_engineering#Text-to-text).

WU, Y.; HU, N.; QI, G.; BI, S.; REN, J.; XIE, A.; SONG, W. Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering. **arXiv preprint arXiv:2309.11206**, 2023.

WU, Z.; PAN, S.; CHEN, F.; LONG, G.; ZHANG, C.; PHILIP, S. Y. A comprehensive survey on graph neural networks. **IEEE transactions on neural networks and learning systems**, IEEE, v. 32, n. 1, p. 4–24, 2020.

XIE, Y.; ALLAIRE, J. J.; GROLEMUND, G. **R markdown: The definitive guide**. [S. l.]: Chapman and Hall/CRC, 2018.

YANI, M.; KRISNADHI, A. A. Challenges, techniques, and trends of simple knowledge graph question answering: a survey. **Information**, MDPI, v. 12, n. 7, p. 271, 2021.