UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTÁVIO CALAÇA XAVIER

ARANDU: *Framework* para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[ ] Dissertação [X] Tese [ ] Outro*:_____

*No caso de mestrado/doutorado profissional, indique o formato do Trabalho de Conclusão de Curso, permitido no documento de área, correspondente ao programa de pós-graduação, orientado pela legislação vigente da CAPES.

Exemplos: Estudo de caso ou Revisão sistemática ou outros formatos.

2. Nome completo do autor

Otávio Calaça Xavier

3. Título do trabalho

ARANDU: Framework para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Anderson Da Silva Soares**, **Professor do Magistério Superior**, em 21/11/2025, às 18:31, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Otavio Calaca Xavier**, **Professor do Magistério Superior**, em 25/11/2025, às 17:58, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Otavio Calaca Xavier**, **Discente**, em 02/12/2025, às 18:48, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5802736** e o código CRC **4B7FD3B9**.

Referência: Processo nº 23070.050923/2025-41 SEI nº 5802736

OTÁVIO CALAÇA XAVIER

ARANDU: *Framework* para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica

Tese apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação.
Linha de pesquisa: Sistemas Inteligentes e Aplicações.

Orientador: Prof. Dr. Anderson da Silva Soares

Goiânia
2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Xavier, Otávio Calaça
ARANDU: Framework para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica [manuscrito] / Otávio Calaça Xavier. - 2025.
190 f.: il.

Orientador: Prof. Anderson da Silva Soares.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2025.
Bibliografia.
Inclui siglas, algoritmos, lista de figuras, lista de tabelas.

1. Geração Aumentada por Recuperação (RAG). 2. Grandes Modelos de Linguagem (LLM). 3. Grafos de Conhecimento. 4. IA Neuro Simbólica. 5. Resposta a Perguntas. I. Soares, Anderson da Silva, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE TESE

Ata nº **24** da sessão de Defesa de Tese de **Otávio Calaça Xavier**, que confere o título de Doutor em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos vinte e dois dias do mês de outubro de dois mil e vinte e cinco, a partir das oito horas, sala 150 INF, realizou-se a sessão pública de Defesa de Tese intitulada **"ARANDU: Framework para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Anderson da Silva Soares (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Renato de Freitas Bulcão Neto (INF/UFG), membro titular interno; Professor Doutor Thierson Couto Rosa (INF/UFG), membro titular interno; Professor Doutor Cedric Luiz de Carvalho (INF/UFG), membro titular externo; e Gustavo de Assis Costa (IFG/GO), membro titular externo. A realização da banca ocorreu por meio de videoconferência. Durante a arguição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Tese, tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Anderson da Silva Soares, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos vinte e dois dias do mês de outubro de dois mil e vinte e cinco.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Anderson Da Silva Soares**, **Professor do Magistério Superior**, em 22/10/2025, às 11:07, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Renato De Freitas Bulcao Neto**, **Professor do Magistério Superior**, em 22/10/2025, às 11:08, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 22/10/2025, às 11:08, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Cedric Luiz De Carvalho**, **Professor do Magistério Superior**, em 22/10/2025, às 11:08, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gustavo de Assis Costa**, **Usuário Externo**, em 22/10/2025, às 11:38, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Otavio Calaca Xavier**, **Professor do Magistério Superior**, em 23/10/2025, às 19:25, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Otavio Calaca Xavier**, **Discente**, em 02/12/2025, às 18:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5731072** e o código CRC **26695074**.

Referência: Processo nº 23070.050923/2025-41 SEI nº 5731072

OTÁVIO CALAÇA XAVIER

ARANDU: *Framework* para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica

Tese defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Doutor em Ciência da Computação, Banca Examinadora constituída pelos professores:

Prof. Dr. Anderson da Silva Soares
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Renato de Freitas Bulcão Neto
Instituto de Informática – UFG

Prof. Dr. Thierson Couto Rosa
Instituto de Informática – UFG

Prof. Dr. Cedric Luiz de Carvalho
Instituto de Informática – UFG

Prof. Dr. Gustavo de Assis Costa
Campus Jataí – IFG

Otávio Calaça Xavier

Professor na UFG e no IFG e coordenador de projetos de inovação no CEIA. Doutorando em Ciência da Computação, com foco em *Graph Neural Networks* e *Retrieval-Augmented Generation (RAG)*. Possui mestrado em Ciência da Computação pela UFG, obtido em 2011, com ênfase em Web Semântica e Serviços Web. Com mais de 20 anos de experiência em Desenvolvimento de Software e Bancos de Dados, além de mais de 15 anos em Arquitetura de Software e Liderança de Equipes. Desde 2007, atuou como palestrante em mais de 100 eventos nacionais e internacionais. Leciona há mais de 15 anos em cursos de graduação, pós-graduação e capacitação. Atua também como consultor em Aprendizado de Máquina, Ciência de Dados, Engenharia e Arquitetura de Software.

Dedico este trabalho à minha esposa, Cecília, e meus filhos, Henrique e Vicente, pelo amor, paciência e compreensão.

Agradecimentos

A realização deste trabalho contou com a colaboração e apoio de algumas pessoas e instituições. Para estas, deixo os meus sinceros agradecimentos:

Primeiramente, a Deus, pela força e sabedoria concedidas durante esta jornada acadêmica.

À minha mãe, Maria Vitória, por toda dedicação e garra que me inspiraram a seguir na carreira de professor.

Ao meu pai, Waldir, pelos ensinamentos e por me criar passando os valores que me guiaram pela vida.

À minha esposa, Cecília, pelo amor, motivação e apoio incondicional durante toda esta difícil jornada.

Aos meus filhos, Henrique e Vicente, pela alegria e motivação que trazem à minha vida, lembrando-me constantemente da importância de buscar sempre o melhor.

Ao professor Anderson da Silva Soares, pela orientação, conselhos, sugestões, atenção e críticas construtivas.

Ao CEIA (Centro de Excelência em Inteligência Artificial), pelas oportunidades oferecidas e portas abertas para o desenvolvimento deste trabalho.

Ao IFG (Instituto Federal de Goiás), pelo suporte financeiro durante o período de realização deste trabalho.

À todos os amigos e familiares que me apoiaram e colaboraram de alguma forma para a concretização deste trabalho.

Onde está a sabedoria que perdemos no conhecimento? Onde está o conhecimento que perdemos na informação?

T. S. Eliot,
poeta, dramaturgo e crítico literário, laureado com o Prêmio Nobel de Literatura.

Resumo

Xavier, Otávio Calaça. **ARANDU: *Framework* para Geração Aumentada por Recuperação em Grafos de Conhecimento com Fundamentação Neuro-Simbólica**. Goiânia, 2025. 191p. Tese de Doutorado. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

Este trabalho aborda o desafio de Resposta a Perguntas sobre Grafos de Conhecimento (KGQA), um campo transformado pela ascensão dos Grandes Modelos de Linguagem (LLMs), mas que ainda enfrenta limitações como a geração de informações factualmente inconsistentes ("alucinações") e a dificuldade em realizar raciocínios complexos. O objetivo central da pesquisa foi desenvolver e validar uma arquitetura neuro-simbólica que mitigasse as limitações de sistemas de Geração Aumentada por Recuperação (RAG) contemporâneos, propondo-se a resolver de forma integrada os desafios de (1) recuperação de evidências com baixa precisão e cobertura, (2) perda de contexto estrutural na comunicação com o LLM, e (3) ausência de orquestração lógica explícita no processo de raciocínio. Para isso, foi projetado, implementado e disponibilizado como código aberto o *framework* ARANDU, que materializa a arquitetura proposta. A metodologia se divide em uma etapa de preparação offline, onde são criados índices híbridos (lexical e semântico) e mineradas regras lógicas a partir do grafo, e um pipeline de execução online em três fases: I) Recuperação Híbrida de Evidências, que extrai um subgrafo coeso combinando recuperação lexical, semântica e estruturada em grafo; II) Orquestração Lógica do Contexto, que enriquece o subgrafo com as regras lógicas e pondera os caminhos de inferência mais relevantes; e III) Representação Neural e Geração, onde uma Rede Neural em Grafo (GNN) codifica o subgrafo em uma representação vetorial (*graph token*) que, junto ao contexto textual, condiciona um LLM compacto a gerar a resposta final. A validação empírica, conduzida nos datasets WebQSP e MetaQA e comparada com baselines como NaiveRAG, GraphRAG e G-Retriever, demonstrou que o ARANDU obteve desempenho superior na maioria dos cenários, especialmente em tarefas de raciocínio multi-salto, com melhorias significativas em métricas de qualidade de ranqueamento, como nDCG@10 e MRR. Os resultados também confirmaram que a representação neural via GNN é mais eficaz que a linearização textual e que a arquitetura se mostra computacionalmente eficiente. A pesquisa conclui que a sinergia entre recuperação otimizada, orquestração lógica e representação neural, como implementada no ARANDU, constitui uma solução robusta e eficaz que aumenta a fidelidade e a precisão das respostas em sistemas KGQA, validando a hipótese central do trabalho.

Palavras–chave

Geração Aumentada por Recuperação (RAG), LLM, Grafos de Conhecimento, IA Neuro-Simbólica, Resposta a Perguntas, Orquestração Lógica

Abstract

Xavier, Otávio Calaça. **ARANDU: Framework for Retrieval-Augmented Generation on Knowledge Graphs with Neuro-Symbolic Foundations**. Goiânia, 2025. 191p. PhD. Thesis. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

This work addresses the challenge of Knowledge Graph Question Answering (KGQA), a field transformed by the rise of Large Language Models (LLMs) but still facing limitations such as the generation of factually inconsistent information ("hallucinations") and difficulty in performing complex reasoning. The central objective of this research was to develop and validate a neuro-symbolic architecture that overcomes the limitations of contemporary Retrieval-Augmented Generation (RAG) systems, aiming to integrally solve the challenges of (1) retrieving evidence with low precision and recall, (2) loss of structural context in communication with the LLM, and (3) the absence of explicit logical orchestration in the reasoning process. To this end, the ARANDU framework was designed, implemented, and made available as open source, materializing the proposed architecture. The methodology is divided into an offline preparation stage, where hybrid indexes (lexical and semantic) are created and logical rules are mined from the graph, and an online execution pipeline with three phases: I) Hybrid Evidence Retrieval, which extracts a cohesive subgraph by combining lexical, semantic, and graph-based structured retrieval; II) Logical Context Orchestration, which enriches the subgraph with logical rules and weights the most relevant inference paths; and III) Neural Representation and Generation, where a Graph Neural Network (GNN) encodes the subgraph into a vector representation (graph token) that, along with the textual context, conditions a compact LLM to generate the final answer. The empirical validation, conducted on the WebQSP and MetaQA datasets and compared with baselines such as NaiveRAG, GraphRAG, and G-Retriever, showed that ARANDU achieved superior performance in most scenarios, especially in multi-hop reasoning tasks, with significant improvements in ranking quality metrics like nDCG@10 and MRR. The results also confirmed that neural representation via GNN is more effective than textual linearization and that the architecture is computationally efficient. The research concludes that the synergy between optimized retrieval, logical orchestration, and neural representation, as implemented in ARANDU, constitutes a robust and effective solution that increases the fidelity and precision of answers in KGQA systems, thus validating the central hypothesis of this work.

Keywords

Retrieval-Augmented Generation (RAG), LLM, Knowledge Graphs, Neuro-Symbolic AI, Question Answering (QA), Logical Orchestration

Lista de Siglas

[AI] *Artificial Intelligence* (Inteligência Artificial)

[AMIE] *Association Rule Mining with Interpolated Explanations* (Mineração de Regras de Associação com Explicações Interpoladas)

[ANN] *Approximate Nearest Neighbors* (Aproximação de Vizinhos Mais Próximos)

[API] *Application Programming Interface* (Interface de Programação de Aplicações)

[ARANDU] *Augmented Retrieval for Advanced Neuro-Symbolic Driven Understanding* (Recuperação Aumentada para Compreensão Neuro-Simbólica Avançada)

[ASCII] *American Standard Code for Information Interchange* (Código Padrão Americano para Intercâmbio de Informações)

[BERT] *Bidirectional Encoder Representations from Transformers* (Representações Bidirecionais de Encoder de Transformers)

[BFS] *Breadth-First Search* (Busca em Largura)

[CPU] *Central Processing Unit* (Unidade Central de Processamento)

[CSV] *Comma-Separated Values* (Valores Separados por Vírgulas)

[DCG] *Discounted Cumulative Gain* (Ganho Acumulado Descontado)

[DPO] *Direct Preference Optimization* (Otimização Direta da Preferência)

[DPR] *Dense Passage Retrieval* (Recuperação de Passagens Densas)

[FAISS] *Facebook AI Similarity Search* (Busca de Similaridade do Facebook)

[FOAF] *Friend of a Friend* (Amigo de um Amigo)

[GAT] *Graph Attention Network* (Rede Neural de Atenção de Grafos)

[GB] *Gigabyte*

[GC] Grafos de Conhecimento

[GCN] *Graph Convolutional Network* (Rede Neural de Convolução de Grafos)

[GGNN] *Graph Neural Network* (Rede Neural de Grafos)

[GNN] *Graph Neural Network* (Rede Neural de Grafos)

[GOFAI] *Good Old-Fashioned AI* (Inteligência Artificial Clássica)

[GPT] *Generative Pre-trained Transformer* (Transformador Pré-treinado Generativo)

[GPU] *Graphics Processing Unit* (Unidade de Processamento Gráfico)

[GQL] *Graph Query Language* (Linguagem de Consulta de Grafos)

[HNSW] *Hierarchical Navigable Small World* (Hierarquias de Vizinhos Próximos Pequenos Mundos)

[IA] Inteligência Artificial

[IDCG] *Ideal Discounted Cumulative Gain* (Ganho Acumulado Descontado Ideal)

[IDF] *Inverse Document Frequency* (Frequência Inversa do Documento)

[IEC] *Information Retrieval Evaluation* (Avaliação de Recuperação de Informações)

[IFG] Instituto Federal de Goiás

[IRI] *Internationalized Resource Identifier* (Identificador de Recurso Internacionalizado)

[ISO] *International Organization for Standardization* (Organização Internacional de Normalização)

[JSON] *JavaScript Object Notation* (Notação de Objeto JavaScript)

[KG] *Knowledge Graph* (Grafo de Conhecimento)

[KGP] *Knowledge Graph Prompting* (Prompting de Grafo de Conhecimento)

[KGQA] *Knowledge Graph Question Answering* (Resposta a Perguntas sobre Grafos de Conhecimento)

[LLM] *Large Language Model* (Grande Modelo de Linguagem)

[LPG] *Labelled Property Graph* (Grafo de Propriedades Rotulado)

[LSTM] *Long Short-Term Memory*

[MLP] *Multi-Layer Perceptron* (Perceptron de Múltiplas Camadas)

[MMLU] *Massive Multitask Language Understanding*

[MRR] *Mean Reciprocal Rank* (Média do Ranking Recíproco)

[NLP] *Natural Language Processing* (Processamento de Linguagem Natural)

[OWL] *Ontology Web Language* (Linguagem de Ontologia Web)

[PCA] *Principal Component Analysis* (Análise de Componentes Principais)

[PCST] *Prize-Collecting Steiner Tree* (Árvore de Steiner de Prêmios)

[PGQL] *Property Graph Query Language* (Linguagem de Consulta de Propriedades de Grafos)

[PLN] Processamento de Linguagem Natural

[PRF] *Probabilistic Relevance Framework* (Framework de Relevância Probabilística)

[QA] *Question Answering* (Resposta a Perguntas)

[RAG] *Retrieval-Augmented Generation* (Geração Aumentada por Recuperação)

[RDF] *Resource Description Framework* (Framework de Descrição de Recursos)

[RDFS] *RDF Schema* (Esquema RDF)

[RI] *Recuperação de Informações*

[RLHF] *Reinforcement Learning from Human Feedback* (Aprendizado por Reforço com Feedback Humano)

[RRF] *Reciprocal Rank Fusion* (Fusão de Ranking Recíproco)

[RTX] *Ray Tracing X*

[SFT] *Supervised Fine-Tuning* (Ajuste Fino Supervisionado)

[SPARQL] *SPARQL Protocol and RDF Query Language*

[SQL] *Structured Query Language*

[SURGE] *SUbgraph Retrieval-augmented GEneration*

[UFG] Universidade Federal de Goiás

[URI] *Uniform Resource Identifier* (Identificador de Recurso Uniforme)

[VRAM] *Video RAM* (Memória de Vídeo)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

1.1 Motivação

A ascensão dos Grandes Modelos de Linguagem (do inglês, *Large Language Models* - LLMs) representa um ponto de inflexão na Inteligência Artificial (IA) contemporânea, redefinindo o estado da arte em um vasto espectro de tarefas de Processamento de Linguagem Natural (PLN) [Brown et al. 2020]. Dotados de uma capacidade sem precedentes para compreender e gerar texto com notável fluência, esses modelos demonstraram particular eficácia em domínios complexos como o de Resposta a Perguntas (*Question Answering* - QA). No entanto, o conhecimento internalizado por esses modelos é, por natureza, **estático**, um reflexo do corpus de treinamento em um ponto específico no tempo, e **não rastreável**, o que dificulta a verificação de suas fontes. Para superar tais deficiências, a Geração Aumentada por Recuperação (RAG) emergiu como um paradigma dominante [Lewis et al. 2020], integrando fontes de conhecimento externas para permitir respostas mais precisas e relevantes. Tal capacidade estabelece essa classe de sistemas como a tecnologia fundamental para o desenvolvimento de assistentes virtuais avançados e motores de busca de nova geração [Gao et al. 2023].

Apesar desse avanço notável, a crescente adoção de LLMs em aplicações de mundo real expôs desafios persistentes que limitam seu potencial. O mais proeminente destes é a propensão a gerar conteúdo que, embora plausível, não se fundamenta em fatos verificáveis. Este fenômeno, tecnicamente denominado “alucinação”, é extensivamente documentado na literatura [Ji et al. 2023]. Ironicamente, mesmo sistemas RAG, concebidos para mitigar este problema ao fundamentar a geração em evidências externas, permanecem vulneráveis. A recuperação de trechos de texto semanticamente próximos, mas factualmente irrelevantes ou contraditórios, pode poluir o contexto do LLM e induzi-lo ao erro, meramente deslocando o desafio da geração para a etapa de recuperação. A manifestação de alucinações representa uma barreira crítica para a adoção segura de LLMs em domínios de alto risco, como medicina e direito, onde a exatidão factual é imperativa [Huang et al. 2025]. Nestes contextos, uma resposta eloquente, porém incorreta, pode acarretar consequências mais severas do que a ausência de uma resposta.

Paralelamente ao desafio da veracidade, a escalabilidade dos LLMs introduz a questão do custo computacional. Modelos de estado da arte, com contagens de parâmetros na casa das centenas de bilhões, demandam uma infraestrutura de grande porte [The BigScience Workshop 2023]. Uma manifestação prática deste desafio é a janela de contexto finita: a abordagem de "força bruta", que trata o conhecimento como um fluxo de texto linearizado e indiferenciado, não apenas excede frequentemente a capacidade de entrada dos modelos, mas também se mostra inerentemente ineficiente, ignorando as ricas relações estruturais que conectam as informações. Este custo impõe uma barreira substancial ao avanço da pesquisa e à democratização da tecnologia, concentrando o desenvolvimento em um número limitado de entidades com vastos recursos [Maliakel, Ilager e Brandic 2025].

Em contrapartida às limitações dos sistemas puramente neurais, o paradigma simbólico, representado por Grafos de Conhecimento (GCs) e sistemas baseados em lógica, oferece um caminho alternativo para a representação e o raciocínio. Os GCs fornecem representações de conhecimento explícitas, estruturadas e editáveis, constituindo uma estratégia complementar para mitigar as deficiências dos LLMs [Sun et al. 2024]. A principal vantagem dessa abordagem reside na sua **precisão e verificabilidade**: o conhecimento é formalizado em triplas ou axiomas, permitindo consultas determinísticas e a construção de cadeias de raciocínio transparentes e explicáveis. No entanto, sistemas puramente simbólicos historicamente enfrentam seus próprios desafios, notadamente a fragilidade (*brittleness*) ao lidar com a ambiguidade e a variabilidade da linguagem natural, além de dificuldades de escalabilidade na construção e manutenção [Plenz e Frank 2024]. A natureza complementar desses dois paradigmas, a flexibilidade semântica dos LLMs e a precisão estrutural dos GCs, tem impulsionado a pesquisa em direção a arquiteturas híbridas [Ma et al. 2025, Chen et al. 2025, He et al. 2024], que buscam unificar ambos para alcançar uma compreensão de máquina mais robusta e fiel.

Diante deste cenário, delineia-se um desafio fundamental na comunidade de pesquisa: como conciliar o poder de **generalização semântica** do paradigma neural com a **precisão e verificabilidade** do paradigma simbólico? A resposta reside na necessidade de desenvolver sistemas de QA que não apenas herdem a capacidade de conversação dos LLMs, mas que também operem com um alto grau de confiabilidade e eficiência. Surge, assim, a motivação para arquiteturas que garantam a geração de respostas fundamentadas (*grounded*), ou seja, que sejam rastreáveis e verificáveis com base em uma fonte de conhecimento explícita e confiável [Rashkin et al. 2023]. Adicionalmente, é importante que essas arquiteturas sejam computacionalmente eficientes. Ou seja, sistemas genuinamente de baixo custo (*lightweight*), capazes de entregar alto desempenho sem depender de infraestruturas de hardware proibitivas. Esta tese se insere precisamente nesta fronteira de pesquisa, motivada pela busca de um paradigma que harmonize a fluência neural com o

rigor simbólico, visando uma nova geração de sistemas de QA confiáveis e acessíveis.

O reconhecimento dessas forças e fraquezas complementares solidificou a busca por arquiteturas híbridas como uma fronteira promissora da pesquisa em IA. Contudo, frequentemente o raciocínio lógico é delegado inteiramente ao LLM, que pode ignorar as restrições estruturais, ou a recuperação simbólica permanece isolada da compreensão semântica profunda oferecida pelos modelos neurais [Sun et al. 2024, Cao et al. 2025].

É precisamente nesta lacuna que esta tese se insere. O desafio fundamental não é apenas unir os dois paradigmas, mas projetar uma arquitetura onde eles se aprimorem mutuamente, de forma coesa e eficiente. A motivação central, portanto, reside na seguinte questão de pesquisa: é possível projetar uma arquitetura que harmonize a fluência neural com o rigor simbólico, utilizando este último como uma camada de orquestração lógica explícita para garantir respostas fundamentadas e, ao mesmo tempo, manter-se computacionalmente eficiente e acessível?

1.2 Definição do Problema

A tarefa central abordada nesta tese é a de Resposta a Perguntas sobre Grafos de Conhecimento (do inglês, *Knowledge Graph Question Answering* - KGQA). Formalmente, dado uma pergunta em linguagem natural q e um Grafo de Conhecimento $G = (V, E)$, onde V é o conjunto de entidades e E é o conjunto de relações, o objetivo é gerar uma resposta em linguagem natural A. Uma solução ideal para este problema deve satisfazer dois critérios fundamentais: a resposta A deve ser **factualmente correta** de acordo com o conhecimento contido em G e, ao mesmo tempo, **semanticamente coerente** e relevante para a pergunta q. A principal exigência é que a resposta seja explicitamente fundamentada (*grounded*) em um subconjunto de evidências $G_{\text{sub}} \subseteq G$, recuperado a partir do Grafo de Conhecimento G.

Apesar da aparente simplicidade desta formulação, as arquiteturas de RAG contemporâneas enfrentam um desafio duplo que compromete a sua eficácia e confiabilidade. O primeiro reside na etapa de recuperação de evidências. As abordagens convencionais de RAG frequentemente se baseiam em uma busca por **proximidade semântica**, presumindo que textos que discutem os mesmos tópicos da pergunta conterão a resposta.

Entretanto, essa suposição apresenta limitações. A semelhança semântica não assegura correção factual ou relevância lógica, resultando na recuperação de informação ruidosa que compromete a qualidade do contexto fornecido ao LLM [Chen et al. 2025]. Adicionalmente, a recuperação puramente semântica sofre de **baixa cobertura**, falhando em capturar evidências que são estruturalmente essenciais, mas semanticamente distantes da pergunta. Por exemplo, para uma pergunta como “Onde nasceu a esposa de Barack Obama?”, a resposta correta é “Chicago”. No entanto, um sistema de busca vetorial

pode não recuperar "Chicago" diretamente, pois esta entidade não possui alta similaridade semântica com os termos da pergunta. A resposta só pode ser inferida através da travessia de um caminho no grafo: `(Barack Obama, spouse, Michelle Obama)` $\rightarrow$ `(Michelle Obama, place_of_birth, Chicago)`. A entidade "Chicago" é, portanto, estruturalmente relevante, mas semanticamente distante.

Por outro lado, a baixa precisão manifesta-se quando o sistema recupera informações semanticamente similares, mas factualmente irrelevantes. Utilizando o mesmo exemplo sobre "Onde nasceu a esposa de Barack Obama?", o sistema pode recuperar documentos que descrevem a carreira política de Barack Obama, ou biografias de outras primeiras-damas. Embora todos semanticamente relacionados, nenhum contém a resposta específica para a pergunta. A informação irrelevante pode obscurecer os dados pertinentes, desviando o foco do modelo e impactando negativamente a geração subsequente [He et al. 2024]. Este fenômeno caracteriza o **problema da recuperação de baixa precisão**. A precisão quantifica a proporção de documentos relevantes dentro do conjunto total recuperado. Uma baixa precisão resulta na entrega de um contexto com **relação sinal-ruído** inadequada ao LLM: o sinal é obscurecido por ruído. Assim, o objetivo central de um sistema de recuperação eficaz é otimizar essa relação, assegurando que o LLM opere sobre um contexto de alta qualidade.

Para mitigar o problema da recuperação de baixa precisão, uma vertente de pesquisa emergente propõe a Geração Aumentada por Recuperação baseada em Grafos (GraphRAG) [Peng et al. 2024]. Diferentemente da recuperação baseada apenas em similaridade semântica, a abordagem GraphRAG explora a topologia do grafo de conhecimento. A premissa é que a relevância de uma evidência não é determinada apenas por seu conteúdo semântico isolado. Ela também depende de suas conexões com outras entidades e relações no grafo. Ao recuperar subgrafos coesos e estruturalmente informados, em vez de documentos desvinculados, os sistemas GraphRAG buscam fornecer ao LLM um contexto mais rico e interconectado. Isso aumenta a probabilidade de que as evidências corretas estejam presentes.

O segundo desafio abordado nesta tese emerge de uma lacuna latente no processamento do contexto recuperado por sistemas GraphRAG. Uma prática metodológica prevalecente, documentada na literatura [Pan et al. 2024], consiste na **linearização do grafo**: a conversão do subgrafo de evidências G_{sub} em uma sequência de texto plano para ser concatenada ao prompt do LLM. Tal abordagem, embora simplifique a integração, resulta em uma perda da topologia do grafo. Consequentemente, o viés indutivo relacional, uma propriedade inerente à estrutura do conhecimento, não é explicitamente transmitido ao modelo de linguagem. A tarefa de reconstruir e interpretar essa estrutura é, portanto, delegada implicitamente ao LLM, que opera sem garantias sobre a consistência dos caminhos relacionais. Isso define o **problema da representação de contexto estrutural**:

como preservar e transmitir a semântica topológica de um grafo de conhecimento ao LLM de forma eficiente e fiel, para além da serialização textual.

O terceiro desafio fundamental reside na ausência de orquestração lógica explícita durante o processo de geração da resposta. Mesmo quando as evidências corretas são recuperadas e fornecidas no contexto, os LLMs podem gerar respostas não fundamentadas ou logicamente inconsistentes. Sistemas RAG contemporâneos delegam implicitamente ao LLM a responsabilidade de interpretar e raciocinar corretamente sobre as evidências fornecidas, sem implementar mecanismos explícitos de raciocínio lógico.

Esta dependência das capacidades de raciocínio do LLM ignora limitações bem documentadas destes modelos em tarefas de raciocínio dedutivo e manutenção de consistência factual [Ng, Matsuba e Zhang 2025]. A natureza probabilística da geração de texto agrava o problema, uma vez que pode produzir respostas semanticamente coerentes, mas factualmente incorretas. Tal comportamento compromete a confiabilidade do sistema, especialmente em domínios onde a precisão factual é necessária.

Isso define o **problema da orquestração lógica**: a delegação irrestrita do processo de raciocínio ao LLM sobre um conjunto de evidências estático. O problema reside na suposição de que o subgrafo recuperado é não apenas factualmente correto, mas também logicamente completo e coerente para que o modelo deduza uma resposta válida. Esta suposição ignora que o contexto pode conter caminhos de inferência incompletos ou atribuir igual importância a evidências com relevâncias diferentes. A lacuna, portanto, é a ausência de um mecanismo explícito que estruture o raciocínio sobre as evidências recuperadas, deixando o LLM operar sem um guia lógico que valide a suficiência e a pertinência do contexto.

Portanto, o problema central desta tese é superar essa tripla limitação. O problema abordado é: **como projetar uma arquitetura integrada para KGQA que resolva de forma eficiente os desafios de (1) recuperação de baixa precisão e cobertura, (2) representação de contexto estrutural e (3) orquestração lógica do raciocínio.** A solução requer uma sinergia entre recuperação neural, representação de grafos e mecanismos de raciocínio simbólico (lógico).

1.3 Hipótese Central

A hipótese central desta tese é que uma arquitetura neuro-simbólica pode superar as limitações de sistemas RAG convencionais ao resolver de forma integrada os desafios de recuperação, representação e orquestração. **É proposto que, ao combinar sinergicamente: (1) uma recuperação híbrida para maximizar a precisão e cobertura das evidências; (2) uma representação neural estrutural para preservar a topologia do grafo; e (3) uma orquestração lógica explícita para expandir e ponderar o contexto;**

é possível atingir um desempenho de ponta em cobertura, precisão e fidelidade em tarefas de resposta a perguntas em Grafos de Conhecimento, com explicabilidade e eficiência computacional.

O ARANDU (*Augmented Retrieval for Advanced Neuro-Symbolic Driven Understanding*), framework proposto nesta tese, foi projetado como uma instanciação concreta desta hipótese. Ele possui módulos que mapeiam diretamente cada uma das componentes propostas, servindo como a base para a validação empírica dessa hipótese.

1.4 Visão Geral da Arquitetura Proposta

Para validar a hipótese central, foi concebida uma arquitetura de RAG neuro-simbólico, implementada no framework que denominamos **ARANDU**, sigla para *Augmented Retrieval for Advanced Neuro-Symbolic Driven Understanding*, que em português pode ser traduzido como "Recuperação Aumentada para Compreensão Neuro-Simbólica Avançada". O nome ARANDU, além de ser um acrônimo, significa "sabedoria" ou "conhecimento" em tupi-guarani, remetendo à ideia de um sistema fundamentado e inteligente. A arquitetura é organizada em duas etapas principais: Preparação e Execução.

A etapa de preparação é composta por dois módulos: **Indexação Híbrida**, responsável por criar e indexar os artefatos para a recuperação textual e **Enriquecimento Lógico**, responsável pela geração de regras lógicas (e.g., transitividade, simetria), extraídas do grafo de conhecimento por ferramentas de mineração de regras [Lajus, Galárraga e Suchanek 2020].

A etapa de execução é composta por três fases distintas: (I) Recuperação Híbrida de Evidências, (II) Orquestração Lógica do Contexto, e (III) Representação Neural e Geração da Resposta. A seguir, os módulos de cada fase são apresentados.

Fase I - Módulo de Recuperação Textual de Candidatos. O primeiro módulo da **Fase de Recuperação de Evidências** tem como objetivo lidar com a vastidão do conteúdo textual que pode estar relacionado a um grafo de conhecimento. Diante de uma pergunta em linguagem natural, é recuperado um conjunto inicial de entidades e relações candidatas que apresentam alta similaridade semântica com a pergunta. Esta etapa atua selecionando um subconjunto promissor de evidências a partir do qual a recuperação estruturada pode operar de forma mais eficiente.

Fase I - Módulo de Recuperação Estruturada. Ainda na **Fase de Recuperação de Evidências**, o segundo módulo eleva a qualidade do conjunto de evidências. A partir dos candidatos selecionados pelo módulo anterior, seu objetivo é extrair um subgrafo que seja não apenas relevante, mas também estruturalmente coeso e informacionalmente denso. Para isso, a arquitetura emprega algoritmos de otimização em grafos, como o *Prize-Collecting Steiner Tree* (PCST) [He et al. 2024]. Essa abordagem formal trata a

relevância semântica das evidências como "prêmios" e a complexidade do subgrafo como "custos", encontrando uma solução ótima que maximiza o sinal (informação útil) enquanto minimiza o ruído (informação irrelevante), combatendo diretamente o desafio da recuperação de baixa precisão.

Fase II - Módulo de Enriquecimento Lógico. Iniciando a **Fase de Orquestração Lógica do Contexto**, o **Módulo de Enriquecimento Lógico** expande o subgrafo recuperado através de inferência lógica. Este módulo identifica lacunas informacionais no subgrafo e aplica regras lógicas previamente mineradas para inferir triplas ausentes mas logicamente deriváveis. Dessa forma, o módulo combate a incompletude inerente aos grafos de conhecimento, enriquecendo a representação com conhecimento implícito e garantindo maior densidade informacional para o raciocínio subsequente.

Fase II - Módulo de Ponderação de Caminhos. Na sequência, o **Módulo de Ponderação de Caminhos** avalia sistematicamente a relevância das trilhas no grafo enriquecido. Este módulo tem como objetivo central identificar e priorizar caminhos que estabelecem conexões semanticamente coerentes entre as entidades da pergunta e as possíveis respostas. O módulo atribui pesos diferenciados aos caminhos: valores elevados para trilhas que demonstram alinhamento semântico com a pergunta, e valores reduzidos para conexões que apresentam inconsistências ou baixa relevância informacional. Esta ponderação sistemática assegura que apenas evidências com maior qualidade relacional sejam priorizadas no processo de raciocínio, combatendo diretamente o problema de ruído informacional em grafos de conhecimento densos.

Fase III - Módulo de Representação Neural via GNN. Este módulo inicia a **Fase de Representação Neural e Geração da Resposta**. Assim, servindo como a ponte entre a evidência estruturada e já validada e o LLM. O subgrafo, otimizado pelo módulo de recuperação estruturada e refinado logicamente pelo módulo de orquestração lógica, é processado por uma Rede Neural em Grafo (GNN), como uma *Graph Attention Network* (GAT) [Veličković et al. 2018, Brody, Alon e Yahav 2022]. A GNN transforma a estrutura do grafo em um "*token* de grafo", um *soft prompt* contínuo que é injetado no LLM. Este mecanismo condiciona o modelo de linguagem a respeitar o viés indutivo relacional da evidência, guiando seu mecanismo de atenção para raciocinar sobre as conexões estruturais apresentadas [He et al. 2024].

Fase III - Módulo de Geração da Textual. Finalmente, na **Fase de Geração da Resposta**, as evidências estruturadas, aumentadas e ponderadas são apresentadas a um LLM de base compacto (e.g., Llama 3.1 8B) para a síntese da resposta final. A arquitetura alinha-se, assim, à pesquisa sobre a eficácia de modelos menores quando aumentados por estruturas de conhecimento robustas [Abdin et al. 2024].

1.5 Contribuições

Este trabalho de tese apresenta um conjunto de contribuições originais para o campo de Resposta a Perguntas sobre Grafos de Conhecimento (KGQA), com ênfase na sinergia entre abordagens neurais e simbólicas para alcançar alta fidelidade e eficiência. Nossas principais contribuições são:

- **O Projeto de uma Nova Arquitetura Neuro-Simbólica para RAG:** Propõe-se uma arquitetura que integra de forma sinérgica recuperação textual e estruturada com orquestração lógica e representação neural para sistemas de geração aumentada por recuperação (RAG). A contribuição central é a harmonização da expressividade semântica de modelos neurais com a precisão formal do raciocínio simbólico, resultando em um sistema que supera as limitações de abordagens puramente neurais ou simbólicas isoladas.
- **A Implementação de um Framework de Código Aberto (ARANDU):** A arquitetura proposta é materializada no framework **ARANDU**, uma implementação de referência em código aberto. Este framework não apenas serve como o veículo para a validação empírica desta tese, mas também é uma contribuição prática para a comunidade, promovendo a reprodutibilidade e permitindo futuras pesquisas na área.
- **Uma Metodologia de Orquestração Lógica para RAG:** Foi introduzida uma fase de orquestração lógica explícita no processo RAG. Nela, é realizada a materialização de regras lógicas mineradas automaticamente e a aplicação delas durante a fase de recuperação para expandir e ponderar caminhos do subgrafo recuperado. Com isso, demonstramos um aumento significativo na fidelidade e na precisão das respostas, mitigando a geração de informações inconsistentes ou ilógicas.
- **Representação Neural de Contexto Logicamente Ponderado:** Foi desenvolvido um mecanismo de representação que utiliza como entrada o subgrafo ponderado pela fase de orquestração. O subgrafo, agora com suas arestas ponderadas pela relevância lógica dos caminhos dos quais elas fazem parte, é codificado por uma *Graph Attention Network* (GATv2) [Brody, Alon e Yahav 2022]. Esta técnica traduz a importância inferida simbolicamente em um viés indutivo para o modelo neural, garantindo que o LLM receba um contexto que não é apenas estruturado, mas também priorizado logicamente.
- **Análise Abrangente de Desempenho e Eficiência:** Foi realizada uma avaliação empírica detalhada, comparando o ARANDU com baselines de estado da arte em benchmarks consagrados como o WebQSP e o MetaQA. Nossos experimentos não apenas validam os resultados da arquitetura em métricas de acurácia, mas também, através de análises qualitativas detalhadas e estudo de caso. Adicionalmente, comprovamos a viabilidade da arquitetura como uma solução “leve”, apresentando aná-

lises de latência e custo computacional que demonstram sua eficiência em comparação com modelos de grande escala.

1.6 Estrutura da Tese

O restante desta tese está organizado da seguinte forma:

- O **Capítulo 2** estabelece a fundamentação teórica, apresentando os conceitos essenciais sobre Grafos de Conhecimento, Geração Aumentada por Recuperação (RAG), Redes Neurais em Grafos (GNNs) e raciocínio simbólico, que formam a base para o este trabalho.
- O **Capítulo 3** revisa a literatura sobre a evolução dos sistemas RAG, analisando criticamente as estratégias de recuperação e de raciocínio em grafos de conhecimento, para então sintetizar as limitações atuais e identificar a lacuna de pesquisa que esta tese se propõe a preencher.
- O **Capítulo 4** apresenta a arquitetura neuro-simbólica do *framework* ARANDU, detalhando seus módulos conceituais que operam em três fases (Recuperação, Orquestração Lógica e Geração) e descrevendo o processo offline de preparação da base de conhecimento.
- O **Capítulo 5** descreve a implementação de referência do *framework* ARANDU, detalhando os componentes de software, o *stack* tecnológico e os algoritmos que materializam a arquitetura conceitual.
- O **Capítulo 6** detalha a configuração experimental, incluindo as questões de pesquisa, os conjuntos de dados, os sistemas de base para comparação, os detalhes da implementação de referência e as métricas de avaliação.
- O **Capítulo 7** apresenta os resultados dos experimentos, respondendo a cada questão de pesquisa com análises de desempenho, estudos sobre o impacto de cada componente, uma avaliação da eficiência computacional e estudos de caso qualitativos.
- O **Capítulo 8** conclui a tese, recapitulando as contribuições, discutindo as limitações do trabalho e delineando direções para pesquisas futuras.

CAPÍTULO 2

Fundamentação Teórica

A construção de sistemas de Resposta a Perguntas (QA) que sejam simultaneamente fiéis e eficientes demanda a integração de conceitos oriundos de múltiplas subáreas da Inteligência Artificial. Este capítulo estabelece a fundamentação teórica indispensável para a compreensão da arquitetura neuro-simbólica proposta nesta tese. O objetivo é apresentar, de forma autocontida e didática, os pilares conceituais que sustentam o design e a implementação do *framework* ARANDU.

A exposição está organizada em torno de cinco áreas centrais. Primeiramente, são abordadas as técnicas de **Representação de Conhecimento**, com ênfase em Grafos de Conhecimento como formalismo para a estruturação de informações. Em seguida, são revisados os fundamentos da **Recuperação de Informação**, detalhando métodos para a busca de evidências relevantes. A Seção subsequente explora o paradigma de **Geração Aumentada por Recuperação (RAG)**, um componente central da arquitetura. Subsequentemente, são apresentadas as **Redes Neurais em Grafos (GNNs)** como mecanismo para o processamento de dados estruturados. Por fim, discute-se o papel do **Raciocínio Simbólico** na validação da consistência lógica das evidências. Juntos, estes componentes formam o alicerce sobre o qual a contribuição desta tese é construída.

2.1 Representação de Conhecimento em Grafos

A representação de conhecimento é uma área central da Inteligência Artificial. Estuda a formalização do conhecimento sobre o mundo, de modo que sistemas computacionais possam utilizá-lo para resolver tarefas complexas [Brachman e Levesque 2004]. Entre os diversos formalismos propostos, os **Grafos de Conhecimento (GCs)** consolidaram-se como uma abordagem de grande relevância. Capturam e organizam informações estruturadas, descrevendo entidades e as relações entre elas em um formato de grafo [Hogan et al. 2021]. Esta seção detalha os conceitos fundamentais sobre GCs. Aborda seus modelos de dados, linguagens de consulta e os processos de construção e armazenamento.

2.1.1 Grafos de Conhecimento (GCs)

Um grafo é formalmente definido como uma estrutura $G = (V, E)$, onde V representa o conjunto de vértices (ou nós) e E o conjunto de arestas que conectam pares de vértices. Em um multigrafo direcionado e rotulado, múltiplas arestas podem existir entre os mesmos vértices, cada aresta possui uma direção definida, e tanto vértices quanto arestas podem ser associados a rótulos [Diestel 2017].

Um Grafo de Conhecimento (GC) constitui um tipo especializado de multigrafo direcionado e rotulado. Geralmente, ele representa uma rede de entidades do mundo real – tais como pessoas, lugares e conceitos – e as relações entre elas. Formalmente, um GC pode ser definido como $KG = (E, R, T)$, onde E é o conjunto de entidades, R o conjunto de tipos de relações, e T o conjunto de triplas que descrevem os fatos [Hogan et al. 2021]. A unidade fundamental de informação em muitos GCs é a tripla, no formato `(sujeito, predicado, objeto)`, que descreve uma única declaração fatual. Por exemplo, a tripla `(UFG, localizada_em, Goiânia)` codifica o fato de que a Universidade Federal de Goiás está localizada na cidade de Goiânia.

É importante destacar que os GCs podem incorporar informações adicionais através de **atributos**. Estes atributos podem ser associados tanto aos nós (entidades) quanto às arestas (relações). Por exemplo, um nó representando uma pessoa pode conter atributos como `nome`, `data_nascimento` e `nacionalidade`. Similarmente, uma aresta representando uma relação de emprego pode incluir atributos como `data_inicio`, `cargo` e `salario`. Esta capacidade de enriquecimento permite que os GCs capturem não apenas a estrutura das relações, mas também os detalhes contextuais associados a entidades e relacionamentos específicos.

2.1.2 Modelos de Dados em GCs

Existem dois modelos de dados principais para grafos de conhecimento. Cada um possui características próprias e um conjunto específico de tecnologias associadas. Os dois modelos mais utilizados são o **Labelled Property Graph (LPG)** e o **Resource Description Framework (RDF)**.

Labelled Property Graph (LPG)

O modelo *Labelled Property Graph*, ou grafo de propriedades rotulado, organiza a informação em vértices (nós) e arestas (relações). Cada vértice ou aresta pode ter um ou mais rótulos (*labels*) e propriedades adicionais, representadas em pares *chave-valor*. Essa abordagem torna o LPG altamente flexível e intuitivo para modelar dados complexos.

Por exemplo, em um grafo que representa filmes, um vértice do tipo *Ator* pode conter propriedades como nome, data de nascimento e nacionalidade. Uma aresta do

tipo *atuou_em* conectando um ator a um filme pode trazer propriedades como o ano de participação ou o papel desempenhado.

Essa estrutura favorece aplicações práticas em sistemas que precisam lidar com dados heterogêneos e mutáveis. Entre as tecnologias mais conhecidas que utilizam o LPG estão o **Neo4j**, o **Amazon Neptune** (em modo LPG) e o **TigerGraph**.

A Figura 2.1 apresenta um exemplo de grafo de propriedades. Na figura é possivel observar um grafo de propriedades com os rótulos *Movie* e *Person* para os vértices. Para as arestas, tem-se os rótulos *ACTED_IN*, *DIRECTED* e *WROTE*. Além disso, cada vértice e aresta possui propriedades adicionais. No lado direito é possível observar as propriedades *released*, *tagline* e *title* para o vértice selecionado.

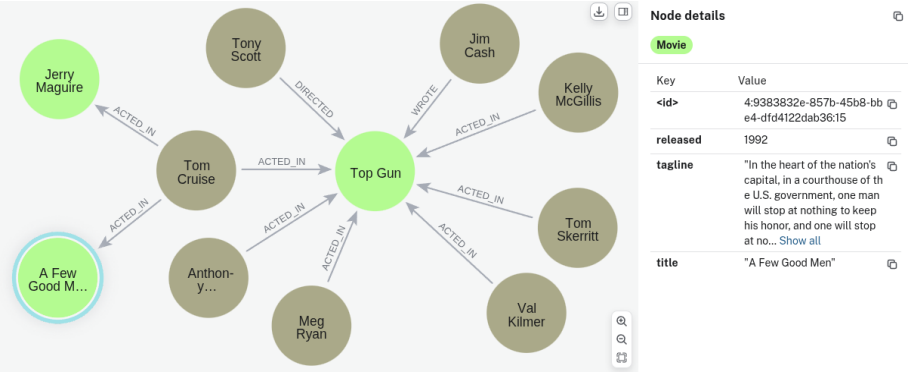

Figura 2.1: Exemplo de grafo de propriedades rotulado (LPG).

Resource Description Framework (RDF)

O *Resource Description Framework* (RDF) é um padrão de modelagem de dados definido pelo *World Wide Web Consortium (W3C)*. Ele foi concebido para representar informações de forma padronizada, interoperável e distribuída na Web.

No RDF, o conhecimento é expresso em **triplas**, compostas por três elementos: **sujeito**, **predicado** e **objeto**. O sujeito representa um recurso (entidade), o predicado descreve uma propriedade ou relação, e o objeto é o valor dessa propriedade, que pode ser outro recurso ou um valor literal.

Por exemplo, a afirmação "*Barack Obama nasceu em Honolulu*" pode ser representada em RDF como a tripla: `(Barack_Obama, nasceuEm, Honolulu)`. Essa formalização permite que diferentes fontes de dados sejam conectadas e interpretadas por máquinas de forma uniforme.

Além da estrutura básica de triplas, o RDF possui extensões como **RDFS (RDF Schema)** e **OWL (Web Ontology Language)**, que permitem descrever hierarquias, restrições e semântica mais rica. As consultas a grafos RDF são realizadas por meio da linguagem **SPARQL**, que se tornou padrão para recuperar e manipular dados nesse modelo.

Entre as tecnologias que implementam repositórios RDF (*RDF triple stores*) destacam-se o **Virtuoso**, o **GraphDB** e o **Blazegraph**.

A Figura 2.2 apresenta um exemplo de grafo com triplas RDF. O exemplo é o mesmo do grafo de propriedades da Figura 2.1. Algumas arestas foram omitidas para melhor visualização. Nota-se que nessa abordagem, todas as propriedades são representadas como triplas. Os rótulos dos vértices são representados como URIs e definidos através do predicado *rdf:type*.

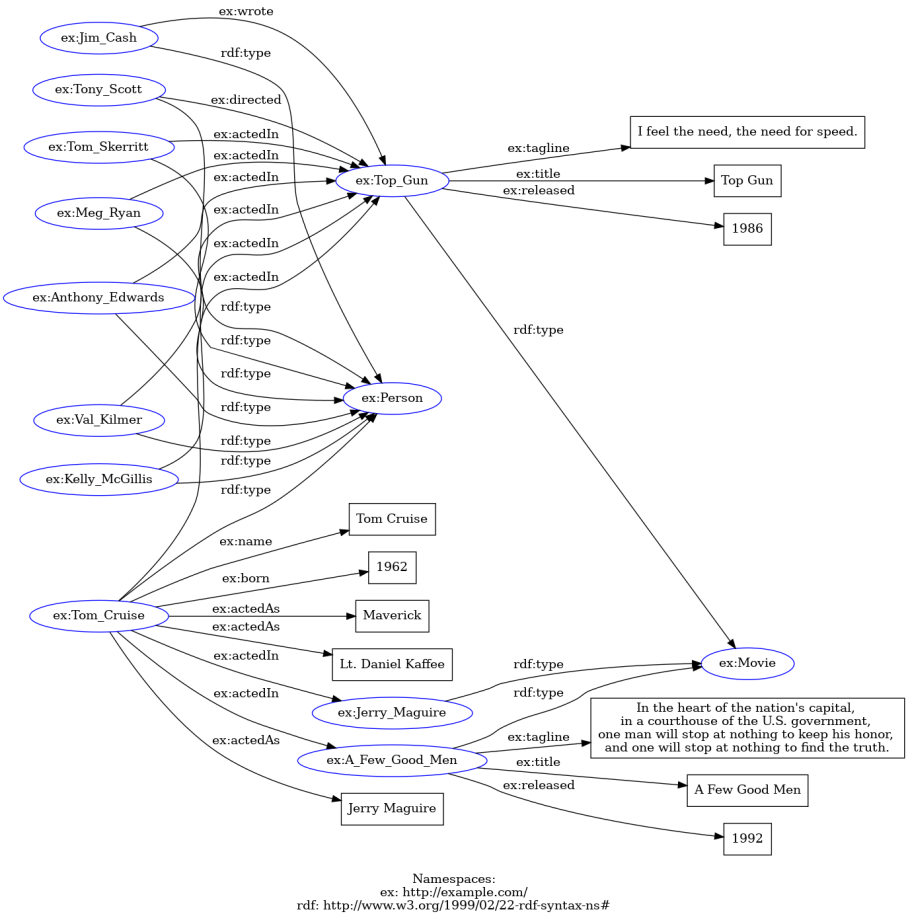

Figura 2.2: Exemplo de grafo com triplas RDF.

Comparação entre LPG e RDF

A tabela 2.1 apresenta uma comparação entre os modelos LPG e RDF. O modelo LPG privilegia flexibilidade e facilidade de uso em aplicações específicas, sendo amplamente adotado em ambientes corporativos e sistemas de recomendação. Já o RDF enfatiza padronização, interoperabilidade e integração de dados em escala global, sendo a base da chamada *Linked Open Data Cloud*[1]. A escolha entre os dois modelos depende do

[1] A Linked Open Data Cloud é uma coleção de conjuntos de dados interligados na web, promovendo a interoperabilidade e o compartilhamento de dados abertos.

contexto da aplicação, dos requisitos de integração e do ecossistema tecnológico adotado.

Tabela 2.1: Comparação entre Labelled Property Graph (LPG) e Resource Description Framework (RDF).

Aspecto	**Labelled Property Graph (LPG)**	**Resource Description Framework (RDF)**
Estrutura	Nós e arestas com rótulos. Propriedades em pares chave-valor.	Triplas `(sujeito, predicado, objeto)`. Baseado em identificadores globais (URIs/IRIs).
Flexibilidade	Estrutura intuitiva e fácil de estender. Bom para dados heterogêneos.	Estrutura rígida, mas altamente padronizada. Facilita a integração de múltiplas fontes.
Padronização	Sem padrão formal universal. Depende da tecnologia usada.	Definido pelo *W3C*. Possui extensões como RDFS e OWL.
Consultas	Linguagens como Cypher (Neo4j) ou GSQL (TigerGraph).	SPARQL como padrão oficial. Ampla adoção em aplicações semânticas.
Tecnologias	Neo4j, Amazon Neptune (modo LPG), TigerGraph.	Virtuoso, GraphDB, Blazegraph, Stardog.
Casos de uso	Sistemas de recomendação, grafos sociais, análise de fraude.	Linked Open Data, integração de dados distribuídos, ontologias e Web Semântica.

O framework ARANDU emprega ambos os modelos: o armazenamento principal é implementado em um Grafo de Propriedades (Neo4j), mas o subgrafo de evidências é temporariamente convertido para RDF para a execução de consultas lógicas com SPARQL.

2.1.3 Linguagens de Consulta em GCs

Para explorar os grafos de conhecimento (GCs), foram desenvolvidas linguagens de consulta específicas. Elas permitem recuperar, filtrar e manipular informações de forma estruturada. Entre as principais linguagens destacam-se o **SPARQL**, associado ao modelo RDF, e o **Cypher**, ligado ao modelo LPG. Mais recentemente, surgiu também o **GQL**, padronizado pela ISO.

SPARQL (SPARQL Protocol and RDF Query Language)

O **SPARQL**[Prud'hommeaux e Seaborne 2008] é a linguagem de consulta padrão para grafos de conhecimento baseados no *Resource Description Framework (RDF)*. Ela foi definida e padronizada pelo *World Wide Web Consortium (W3C)* em 2008, sendo posteriormente ampliada em versões posteriores. Desde então, consolidou-se como a principal linguagem para recuperação, integração e manipulação de dados semânticos.

O SPARQL adota um estilo declarativo, inspirado no *Structured Query Language (SQL)*. Entretanto, em vez de operar sobre tabelas relacionais, as consultas especificam **padrões de grafos** que devem ser correspondidos no repositório RDF. Essa abordagem permite descrever, de forma clara e expressiva, relações entre entidades distribuídas em diferentes fontes de dados.

Um exemplo simples é mostrado no Código 2.1, que recupera todos os filmes e seus respectivos diretores, conforme o grafo ilustrado na Figura 2.2.

Código 2.1 Exemplo de consulta SPARQL para recuperar diretores e filmes

```
PREFIX ex: <http://example.com>
SELECT ?director ?movie
WHERE {
  ?director ex:directed ?movie .
  ?movie ex:title ?title .
}
```

Além das consultas básicas, o SPARQL suporta recursos avançados, como filtros, junções, expressões regulares, operações de agregação e subconsultas. Isso o torna adequado para uma ampla variedade de cenários, desde a exploração simples de dados enciclopédicos até consultas complexas em grandes repositórios científicos e governamentais.

Na prática, o SPARQL é um dos pilares da *Linked Open Data Cloud*. Ele possibilita que diferentes bases RDF sejam interconectadas e acessadas de forma uniforme, garantindo interoperabilidade entre domínios heterogêneos. Sua adoção é ampla tanto no meio acadêmico, em aplicações como *Question Answering* sobre grafos, quanto no meio industrial, em sistemas de integração de dados, análise semântica e inteligência empresarial.

Cypher

O **Cypher**[Neo4j Inc. 2025] é a linguagem de consulta mais difundida para o modelo *Labelled Property Graph (LPG)*. Ela foi originalmente desenvolvida pela empresa **Neo4j**, que a introduziu em 2011 como parte de seu sistema de banco de dados orientado a grafos. Ao longo do tempo, o Cypher tornou-se referência de fato nesse ecossistema,

sendo incorporado em outras plataformas e servindo de inspiração para novas iniciativas de padronização.

A linguagem adota um estilo declarativo, inspirado no *Structured Query Language (SQL)*, mas introduz uma notação própria para grafos. Sua principal característica sintática é a representação visual de padrões de grafos, na qual nós são descritos entre parênteses e arestas por setas direcionadas. Esse recurso facilita a legibilidade das consultas, tornando o Cypher acessível mesmo para usuários sem formação avançada em bancos de dados.

O Código 2.2 ilustra uma consulta em Cypher que recupera todos os atores que atuaram em um filme específico, conforme a estrutura apresentada na Figura 2.1.

Código 2.2 Exemplo de consulta Cypher para recuperar atores que atuaram em um filme específico

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie {title: "Top␣Gun"})
RETURN a.name
```

Além de sua clareza, o Cypher oferece expressividade suficiente para tarefas complexas, como análise de caminhos, descoberta de padrões recorrentes e agregações sobre grandes redes. Por essa razão, ele foi amplamente adotado em aplicações industriais que envolvem sistemas de recomendação, detecção de fraudes, análise de redes sociais e grafos de conhecimento corporativos.

A importância do Cypher também se reflete em sua influência sobre a evolução das linguagens de consulta em grafos. A maior parte de sua sintaxe foi incorporada ao processo de padronização internacional que resultou na **Graph Query Language (GQL)**. Assim, o Cypher pode ser considerado tanto um marco prático quanto um precursor conceitual da atual padronização da área.

GQL (*Graph Query Language*). O ***Graph Query Language (GQL)*** é a primeira linguagem de consulta para grafos padronizada pela ***International Organization for Standardization (ISO)***, publicada oficialmente em 2024 como a norma **ISO/IEC 39075:2024** [ISO/IEC 2024].

A padronização do GQL representa um marco histórico para a área de bancos de dados orientados a grafos. Até então, diferentes linguagens haviam sido propostas e utilizadas, como o **Cypher** (Neo4j), o **PGQL** (Oracle) e o **G-CORE**[Angles et al. 2018] (especificação acadêmica). Essas linguagens compartilhavam princípios semelhantes, mas divergiam em aspectos sintáticos e semânticos, o que dificultava a interoperabilidade entre sistemas.

O GQL foi concebido como sucessor e generalização dessas iniciativas. Tendo como objetivo estabelecer um padrão global de linguagem para consulta e manipulação

de grafos. Sua especificação formaliza a noção de **padrões de grafos**. O que permite expressar consultas declarativas que identificam estruturas e relacionamentos em bases de dados complexas. Assim como o SQL consolidou a área de bancos de dados relacionais, o GQL busca desempenhar papel equivalente no ecossistema dos grafos.

A norma define elementos fundamentais para consultas em grafos rotulados e propriedades associadas. Com isso, abrangendo tanto consultas analíticas quanto operações de atualização. Dessa forma, o GQL fornece uma base comum para que fornecedores de tecnologia, pesquisadores e desenvolvedores construam soluções interoperáveis e robustas.

A expectativa é que, nos próximos anos, o GQL se torne a linguagem dominante para consultas em grafos de conhecimento. Isso deve ocorrer não apenas pela adoção industrial, mas também pelo impacto acadêmico, facilitando a reprodução de experimentos, a integração de sistemas e a consolidação de boas práticas na área.

2.1.4 Grafos de Conhecimento de Larga Escala

A aplicação prática dos GCs é demonstrada por projetos de grande escala que se tornaram recursos de referência na web e na pesquisa em IA [Hogan et al. 2021].

DBpedia

A **DBpedia** é uma das iniciativas mais relevantes no contexto da Web Semântica. Ela está entre os principais exemplos de **grafos de conhecimento abertos**. Criada em 2007 por [Bizer et al. 2007], a DBpedia surgiu com o objetivo de extrair informações estruturadas da Wikipedia. Esses dados foram disponibilizados em formato *RDF (Resource Description Framework)*.

O processo de transformação possibilitou que os dados enciclopédicos fossem interpretados e manipulados por máquinas. Isso abriu caminho para consultas semânticas e para a integração com outras fontes de dados na *Linked Open Data Cloud*.

Do ponto de vista técnico, a DBpedia realiza a extração de *infoboxes*, categorias e links. Esses elementos estruturados dos artigos da Wikipedia são convertidos em triplas RDF. Assim, descrevem entidades como pessoas, filmes ou cidades, bem como suas relações.

O resultado é uma vasta rede de entidades interconectadas. A DBpedia funciona como ponto central de ligação entre diversos outros grafos abertos. Sua arquitetura foi projetada para garantir interoperabilidade e reuso. Por isso, é um dos principais nós da *Linked Data*[2] publicada na Web.

[2]O termo *Linked Data* refere-se a um conjunto de práticas para publicar e interligar dados na web,

Ao longo dos anos, a DBpedia tornou-se uma infraestrutura de referência. Ela é usada em pesquisas acadêmicas e em aplicações práticas. Entre os exemplos estão tarefas de **Respostas a Perguntas (*Question Answering* - *QA*)**, **Recuperação de Informação Semântica**, **Sistemas de Recomendação** e **aprendizado de representações em grafos (*graph embeddings*)** [Ristoski e Paulheim 2018]. Além disso, a comunidade aberta do projeto garante atualizações frequentes, que acompanham a evolução constante da Wikipedia.

A importância da DBpedia vai além de ser uma fonte de dados estruturados de alta qualidade. Ela também exerceu um papel histórico como catalisadora do movimento *Linked Data*. Mostrou, na prática, como dados distribuídos e heterogêneos podem ser interligados e explorados de forma integrada.

Wikidata

A **Wikidata** é um grafo de conhecimento colaborativo e aberto. Foi lançada em 2012 sob a iniciativa da *Wikimedia Foundation*[3]. O projeto foi formalmente descrito por [Vrandečić e Krötzsch 2014]. Diferentemente da DBpedia, que extrai dados a partir de conteúdos já existentes na Wikipedia, a Wikidata foi projetada desde o início como um repositório estruturado e independente. Nele, voluntários podem inserir, editar e manter dados de forma colaborativa.

Assim como a DBpedia, a Wikidata adota o modelo de grafos RDF. Contudo, organiza suas informações em um esquema próprio baseado em **itens** e **propriedades**. Os itens são entidades identificadas por **IRIs**[4], como pessoas, lugares, eventos ou conceitos. As propriedades representam relacionamentos e atributos. Cada afirmação pode ser enriquecida com qualificadores e referências. Isso permite representar não apenas fatos, mas também o contexto e a proveniência da informação. Essa característica confere à Wikidata uma grande flexibilidade na representação do conhecimento.

Graças à sua natureza multilíngue e ao forte engajamento da comunidade, a Wikidata se tornou um dos maiores projetos colaborativos de dados abertos do mundo. É amplamente utilizada em aplicações de *Question Answering* e enriquecimento semântico de textos. Também é empregada no treinamento de modelos de linguagem e construção de sistemas inteligentes. Além disso, desempenha um papel central no ecossistema Wikimedia. Fornece dados estruturados para a Wikipedia em diferentes idiomas.

utilizando padrões como URIs e RDF para garantir interoperabilidade e acessibilidade.

[3] A Wikimedia Foundation é uma organização sem fins lucrativos que hospeda e desenvolve projetos educacionais livres, incluindo Wikipedia, Wikidata e outros projetos wiki.

[4] IRI (*Internationalized Resource Identifier*) é uma extensão do URI que permite caracteres Unicode. Exemplo: `http://www.wikidata.org/entity/Q5`.

Assim como a DBpedia, a Wikidata também ocupa posição de destaque na *Linked Open Data Cloud*. A diferença está em oferecer um mecanismo de atualização contínua e colaborativa. Isso amplia significativamente a cobertura e a atualidade dos dados disponíveis.

Freebase

A **Freebase** foi uma base de conhecimento criada em 2007 pela empresa Metaweb[5]. Foi descrita formalmente por [Bollacker et al. 2008]. Seu objetivo era oferecer uma plataforma colaborativa para estruturar o conhecimento humano em formato de grafo. Permitia que tanto especialistas quanto usuários comuns contribuíssem com a inserção e edição de dados.

Diferentemente da DBpedia, que dependia da extração automática de dados da Wikipedia, o Freebase apostava em um modelo híbrido. Combinava curadoria humana com técnicas automáticas de integração de dados. O resultado foi uma base altamente conectada. Abrangia milhões de entidades e relações em domínios como música, cinema, geografia e ciência.

Em 2010, a Google adquiriu a Metaweb. Incorporou o Freebase como uma das principais fontes para a construção do *Google Knowledge Graph*. Este foi lançado em 2012. Essa integração marcou um ponto de virada no uso de bases de conhecimento em larga escala. Influenciou diretamente sistemas de busca, assistentes virtuais e aplicações de inteligência artificial.

Embora tenha sido oficialmente descontinuado em 2016, o Freebase desempenhou um papel fundamental. Parte de seus dados foi migrada para a Wikidata. Serviu como inspiração e precursor de iniciativas posteriores na evolução dos grafos de conhecimento. Sua importância histórica está associada à demonstração de que a colaboração em larga escala poderia viabilizar a construção de bases de conhecimento globais e interconectadas.

2.1.5 Implicações para a Arquitetura Proposta

A representação de conhecimento em grafos constitui um componente estruturante da arquitetura neuro-simbólica proposta. Os grafos de conhecimento fornecem o formalismo necessário que habilita o raciocínio simbólico explícito, uma capacidade que distingue esta tese de abordagens puramente neurais. A estrutura do grafo permite que o

[5] A Metaweb foi uma empresa de tecnologia fundada em 2005, especializada no desenvolvimento de plataformas para estruturação colaborativa de conhecimento.

conhecimento seja não apenas armazenado, mas também manipulado, consultado e expandido de forma logicamente coesa.

A discussão sobre os modelos de dados LPG e RDF fundamenta uma decisão estratégica central na arquitetura: a utilização de um modelo híbrido. A adoção do LPG como modelo de persistência primário oferece flexibilidade e desempenho para a modelagem de domínios complexos. Contudo, a conversão transitória de subgrafos de evidências para o padrão RDF viabiliza a aplicação de inferência lógica por meio de consultas SPARQL. Essa abordagem dual permite que o sistema se beneficie tanto da praticidade do LPG quanto da formalidade semântica do RDF.

Essa escolha representa um princípio de projeto fundamental, não apenas um detalhe de implementação. Ela permite que a arquitetura integre a recuperação de fatos estruturados com a aplicação de regras lógicas. O grafo de conhecimento, portanto, transcende a função de um repositório de dados passivo. Ele atua como um ambiente para a execução de operações de raciocínio, onde evidências são conectadas, expandidas e ponderadas antes da interação com o componente neural.

A materialização desses conceitos na arquitetura proposta é detalhada no Capítulo 4, que descreve o fluxo de dados e o papel do grafo de conhecimento nos processos offline e online. As decisões de implementação, incluindo a escolha do modelo de dados híbrido e as tecnologias para armazenamento e consulta, são aprofundadas no Capítulo 5.

2.2 Recuperação de Informação para QA

Sistemas de Resposta a Perguntas (QA) dependem de evidências confiáveis. A etapa de Recuperação de Informação (RI) localiza tais evidências. Em arquiteturas RAG, essa etapa sustenta a geração e reduz alucinações [Lewis et al. 2020]. Seu objetivo é maximizar a cobertura (*recall*) com baixo ruído e baixa latência [Manning, Raghavan e Schütze 2008].

Pode-se destacar três paradigmas principais de RI neste contexto. O primeiro é a recuperação lexical, baseada em termos e representações esparsas. Esta busca por similaridade lexicográfica de termos entre consulta e documentos. Modelos como BM25[Robertson e Zaragoza 2009] são o padrão de fato na prática. O segundo é a recuperação semântica, baseada em significado e vetores densos. Ela usa *embeddings*[6] e busca por similaridade [Reimers e Gurevych 2019, Karpukhin et al. 2020]. O terceiro é a recuperação híbrida, que combina as duas abordagens. Métodos de fusão, como RRF, elevam a robustez do ranqueamento [Cormack, Clarke e Buettcher 2009].

[6] *Embeddings* são representações vetoriais densas que codificam o significado semântico de textos em espaços numéricos de alta dimensionalidade.

Questões operacionais também importam no desenho do pipeline. É preciso decidir como indexar, ranquear e fundir resultados. Índices vetoriais e ANN reduzem latência em alta dimensão [Johnson, Douze e Jégou 2019, Malkov e Yashunin 2020]. As próximas subseções detalham os paradigmas lexical, semântico e híbrido.

2.2.1 Recuperação Lexical: Abordagens Baseadas em Termos

A recuperação lexical usa representações esparsas de texto. Documentos e consultas são vetores de termos ponderados. Um índice invertido[7] organiza as listas de ocorrência por termo. A pontuação depende da sobreposição termo–documento [Manning, Raghavan e Schütze 2008].

O modelo vetorial clássico aplica pesos *tf-idf* (*Term Frequency-Inverse Document Frequency*). *Tf* mede a frequência no documento; *idf* mede a raridade na coleção. A combinação privilegia termos informativos e reduz termos muito comuns. Essa ideia foi consolidada por Salton e Buckley ainda na década de 1980 [Salton e Buckley 1988].

Com o surgimento do **BM25** (*Best Match 25*), ele logo se tornou o método lexical mais usado. A essência do BM25 é uma função de pontuação. Ela classifica documentos com base na relevância para uma consulta de pesquisa. Esse método deriva do *Probabilistic Relevance Framework*[8]. A pontuação do BM25 para um documento D e uma consulta Q contendo os termos $q_1, \ldots, q_n$ é calculada da seguinte forma:

$$\text{BM25}(q,d) = \sum_{t \in q} IDF(t) \cdot \frac{tf_{t,d}(k_1+1)}{tf_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\overline{|d|}}\right)}$$

Na fórmula, $tf_{t,d}$ é a frequência do termo t em d. $|d|$ é o tamanho de d e $\overline{|d|}$ é o tamanho médio da coleção. k_1 controla a saturação do tf e b a normalização por comprimento. Valores típicos são $k_1 \in [1.2, 2.0]$ e $b \approx 0.75$ [Robertson e Zaragoza 2009].

A recuperação lexical oferece benefícios claros, como alta precisão para palavras-chave exatas e expressões técnicas. Apresenta também bom desempenho com nomes próprios, códigos e siglas. Adicionalmente, é um método eficiente e transparente para fins de depuração [Manning, Raghavan e Schütze 2008].

Entretanto, essa abordagem possui limitações conhecidas, sendo o *lexical mismatch* o problema central. Sinônimos e paráfrases não são facilmente recuperados, pois usuários raramente utilizam os mesmos termos da indexação. Por exemplo, uma busca

[7] Estrutura que mapeia cada termo aos documentos que o contêm.

[8] O *Probabilistic Relevance Framework* (PRF) é um modelo formal para a Recuperação de Informação. Ele classifica documentos com base na probabilidade de relevância para a consulta do usuário. O princípio fundamental do PRF (*Probability Ranking Principle*) estabelece que um sistema de recuperação ideal deve classificar os documentos em ordem decrescente. A ordem se baseia na sua probabilidade de relevância [Robertson 1977].

por “carro” pode não retornar um documento que utiliza o sinônimo “automóvel”. Este fenômeno foi descrito por [Furnas et al. 1987].

O paradigma lexical, contudo, dispõe de mitigadores para atenuar o problema. Técnicas como normalização, remoção de *stopwords*[9] e *stemming*[10] são aplicadas. A precisão local é aprimorada por meio de busca por frases e proximidade. Expansão de consulta e *pseudo-relevance feedback*[11] são outras estratégias úteis [Carpineto e Romano 2012].

Apesar dessas técnicas, a capacidade de lidar com semântica fina permanece limitada. Relações implícitas e paráfrases complexas frequentemente não são capturadas. Tais casos motivam o uso de recuperação densa e abordagens híbridas, detalhadas a seguir.

2.2.2 Recuperação Semântica: Abordagens Baseadas em Significado

A recuperação semântica representa textos em vetores densos. Cada documento e cada consulta viram um *embedding*. A similaridade é medida no espaço vetorial, por exemplo com cosseno. O objetivo é aproximar itens semanticamente relacionados [Reimers e Gurevych 2019].

Modelos *Sentence-Transformers*[Reimers e Gurevych 2020] são usados para gerar *embeddings*. Esses modelos adaptam arquiteturas como o BERT para gerar representações de sentenças. Uma camada de *pooling* é adicionada para agregar as saídas do modelo em um vetor de dimensão fixa. Isso os torna adequados para busca densa e tarefas de similaridade [Reimers e Gurevych 2019].

Em cenários de QA em domínio aberto, modelos de duplo codificador apresentam desempenho notável. O modelo *Dense Passage Retrieval* (DPR), por exemplo, otimiza codificadores distintos para consultas e documentos. Seu treinamento utiliza pares de passagens positivas e negativas, aplicando uma função de perda contrastiva. Essa abordagem aprimora a capacidade do modelo de recuperar evidências relevantes para responder a perguntas [Karpukhin et al. 2020].

A eficiência da busca densa depende de índices vetoriais otimizados. Estruturas de busca por vizinhos próximos aproximados (ANN)[12] reduzem a latência em espaços de alta dimensão. Algoritmos como FAISS e HNSW estão entre as implementações mais utilizadas para essa finalidade [Johnson, Douze e Jégou 2019, Malkov e Yashunin 2020].

[9] Stopwords são palavras muito frequentes e com baixo valor semântico (e.g., “o”, “a”, “de”), geralmente removidas para otimizar a indexação.

[10] Stemming é o processo de redução de uma palavra ao seu radical, permitindo agrupar variações morfológicas (e.g., “correr”, “correndo”).

[11] *Pseudo-relevance feedback* é uma técnica que assume os primeiros documentos recuperados como relevantes e extrai deles novos termos para expandir a consulta original, refinando a busca.

[12] A busca ANN retorna vizinhos próximos com custo computacional inferior ao da busca exata, sendo útil em larga escala.

A recuperação semântica oferece vantagens significativas. Ela demonstra robustez ao lidar com sinônimos, paráfrases e variações morfológicas. Adicionalmente, é resiliente a ruídos textuais e a pequenas diferenças na formulação da consulta. Seu desempenho é superior quando a intenção do usuário não está explícita nos termos da busca [Reimers e Gurevych 2019].

Contudo, a abordagem apresenta limitações importantes. Termos raros, nomes próprios e outras entidades específicas podem receber peso insuficiente durante a codificação. Esse efeito prejudica o desempenho em consultas que demandam correspondência exata de palavras-chave. Por exemplo, uma consulta por um código de erro específico, como "ERR-5432", pode não ser bem interpretada semanticamente, enquanto a recuperação lexical a trataria com exatidão. Além disso, a generalização para domínios distintos dos dados de treinamento pode ser um desafio [Karpukhin et al. 2020].

Existem, no entanto, estratégias de mitigação aplicadas na prática. Uma técnica comum é o uso de *re-rankers*[13] para refinar a lista de candidatos iniciais. O ajuste do índice com filtros baseados em metadados também contribui para aprimorar a precisão. Tais estratégias ajudam a reduzir erros semânticos e a elevar a precisão geral do sistema.

A recuperação densa amplia a cobertura semântica para além da correspondência lexical. Contudo, ela não substitui completamente a precisão oferecida pelos sinais lexicais. Essa complementaridade motiva o desenvolvimento de sistemas de recuperação híbrida. A próxima subseção explora técnicas que integram os resultados de ambos os paradigmas.

2.2.3 Recuperação Híbrida: Unindo o Lexical e o Semântico

A recuperação híbrida surge da complementaridade entre as abordagens lexical e semântica. Enquanto a busca lexical oferece precisão para termos exatos, a busca semântica captura o significado. Nenhuma das duas, isoladamente, consegue lidar com a complexidade de todas as consultas de forma robusta. A fusão de ambos os paradigmas visa a combinar a precisão do primeiro com a abrangência do segundo [Cormack, Clarke e Buettcher 2009].

A combinação dos resultados é geralmente feita por meio de técnicas de fusão. O *Reciprocal Rank Fusion* (RRF)[Cormack, Clarke e Buettcher 2009] é um método de fusão que se destaca pela sua simplicidade e eficácia. Ele combina múltiplas listas de resultados com base na posição dos documentos, e não em seus escores. Essa característica o torna resiliente a diferentes escalas de pontuação dos sistemas de recuperação. A

[13] *Re-rankers* são modelos de segundo estágio que reordenam a lista de candidatos iniciais para refinar a precisão.

pontuação de um documento d no RRF é dada por:

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + \text{rank}_r(d)}$$

onde R é o conjunto de listas de classificação, $\text{rank}_r(d)$ é a posição de d na lista r e k é uma constante de amortização, tipicamente definida como 60.

A principal vantagem da recuperação híbrida é a sua robustez. Ao agregar sinais de relevância distintos, o sistema se torna menos suscetível a falhas individuais. A abordagem tende a melhorar a cobertura (*recall*), recuperando um conjunto mais completo de documentos relevantes. Como resultado, a qualidade das evidências para sistemas de QA e outras tarefas de NLP pode ser aprimorada.

2.2.4 Implicações para a Arquitetura Proposta

A recuperação de informação constitui um módulo fundamental na arquitetura neuro-simbólica proposta. A qualidade das evidências recuperadas determina o potencial das etapas subsequentes de raciocínio e geração. Ela é essencial para mitigar alucinações e garantir a fidelidade factual das respostas. A análise dos paradigmas de recuperação, portanto, fundamenta a adoção de uma abordagem híbrida como a estratégia mais robusta para a arquitetura.

Domínios de conhecimento complexos exigem tanto a precisão da busca lexical para entidades nomeadas e termos técnicos, quanto a flexibilidade da busca semântica para capturar a intenção do usuário. Uma abordagem híbrida pode trazer completude para senários em que uma abordagem exclusivamente lexical ou semântica ser insuficiente. Essa escolha estratégica, portanto, transcende a mera otimização da recuperação de dados.

A recuperação híbrida estabelece uma base sólida sobre a qual o raciocínio lógico pode operar de forma eficaz. A qualidade desta etapa inicial afeta diretamente a capacidade do sistema de construir um conjunto suficientemente rico de evidências para a geração da resposta.

A aplicação destes princípios na arquitetura é detalhada no Capítulo 4, que descreve o módulo de Recuperação Textual como a primeira etapa do pipeline online. As decisões de implementação, incluindo a adoção da estratégia híbrida com fusão RRF e as tecnologias de armazenamento, são aprofundadas no Capítulo 5 (Seção 5.4.1).

2.3 Arquiteturas de Redes Neurais Relevantes

A estruturação do conhecimento e a recuperação de informação são etapas preparatórias. Elas fornecem as evidências para a tarefa de Resposta a Perguntas (QA). Con-

tudo, a interpretação semântica dessas evidências e a síntese da resposta final demandam modelos computacionais avançados. Nesse contexto, as arquiteturas de redes neurais profundas emergiram. Elas redefiniram o estado da arte no Processamento de Linguagem Natural (PLN). Tais arquiteturas também são importantes na área de representação de dados estruturados em grafos.

Esta seção detalha as arquiteturas neurais centrais para a presente tese. Primeiramente, aborda-se a arquitetura Transformer, base dos modelos de linguagem modernos. Em seguida, são definidos os Grandes Modelos de Linguagem (LLMs), com suas capacidades e limitações inerentes. Por fim, são apresentadas as Redes Neurais em Grafos (GNNs), especializadas no processamento de dados não-euclidianos.

Explica a tecnologia base dos LLMs.

2.3.1 O Mecanismo de Atenção e a Arquitetura Transformer

O processamento de linguagem natural (PLN) passou por uma profunda transformação nas últimas décadas. Arquiteturas recorrentes (RNNs) [Elman 1990] e suas variantes, como a *Long Short-Term Memory* (LSTM)[14] [Hochreiter e Schmidhuber 1997], processavam os dados sequencialmente.

Elas obtiveram sucesso em tarefas como tradução automática e modelagem de linguagem. Contudo, apresentavam limitações no tratamento de dependências de longo prazo e na paralelização do processamento. A representação de uma sequência inteira era comprimida em um vetor de contexto de tamanho fixo. Tal vetor limitava a capacidade do modelo de reter informação de longo prazo [Bahdanau, Cho e Bengio 2016].

Para superar essa limitação, foi introduzido o mecanismo de atenção (*attention mechanism*). A atenção permite que um modelo pondere dinamicamente a importância de diferentes partes da sequência de entrada ao produzir uma saída. Em vez de depender de um único vetor de contexto, o modelo aprende a "prestar atenção" a *tokens* específicos da entrada que são mais relevantes para cada passo da saída. Isso se mostrou crucial em tarefas como a tradução automática, onde o alinhamento entre palavras em diferentes idiomas é fundamental [Bahdanau, Cho e Bengio 2016].

A consolidação dessa ideia ocorreu com a publicação do trabalho "*Attention Is All You Need*" [Vaswani et al. 2017]. Os autores propuseram uma nova arquitetura, o Transformer, que dispensava completamente a recorrência. O Transformer baseia-se exclusivamente no mecanismo de atenção para capturar as dependências entre os *tokens* de entrada e saída. Sua arquitetura canônica é composta por um codificador (*encoder*) e

[14] LSTM é a sigla para *Long Short-Term Memory* e trata-se de uma arquitetura de rede neural em que o neurônio é retroalimentado com sua saída, possuindo três portas (entrada, saída e esquecimento) para melhor memorização dos vínculos em amostras sequenciais.

um decodificador (*decoder*). O codificador constrói uma representação rica da sequência de entrada. O decodificador, por sua vez, utiliza essa representação para gerar a sequência de saída.

Uma inovação central do Transformer é a atenção multi-cabeça (*multi-head attention*). Em vez de aplicar a atenção uma única vez, o modelo o faz múltiplas vezes em paralelo. Cada "cabeça" de atenção aprende a focar em diferentes aspectos da sequência. As saídas dessas cabeças são então concatenadas e projetadas, permitindo que o modelo capture um conjunto mais rico e diversificado de relações entre os *tokens* [Vaswani et al. 2017]. A capacidade de processar todos os tokens de uma sequência simultaneamente, em vez de sequencialmente, conferiu ao Transformer um alto grau de paralelização. Essa eficiência computacional foi o fator que viabilizou o treinamento de modelos em uma escala sem precedentes, estabelecendo a fundação para a era dos Grandes Modelos de Linguagem (LLMs).

2.3.2 Grandes Modelos de Linguagem (LLMs)

Um modelo de linguagem é, fundamentalmente, uma distribuição de probabilidade sobre sequências de palavras [Jurafsky e Martin 2023]. Sua tarefa principal é calcular a probabilidade de uma dada sequência de texto ocorrer, o que permite prever a palavra seguinte mais provável. As primeiras abordagens para esta tarefa foram dominadas por modelos estatísticos, como os n-gramas, baseados nos fundamentos da teoria da informação [Shannon 1948]. Apesar de sua eficácia inicial, esses modelos sofriam com a "maldição da dimensionalidade" e a esparsidade de dados, tornando-os limitados para capturar dependências de longo prazo. A "maldição da dimensionalidade" refere-se ao fenômeno em que o espaço de parâmetros cresce exponencialmente com o tamanho do vocabulário. Isso torna necessário um volume de dados de treinamento proibitivamente grande para se obter estimativas estatisticamente confiáveis. A esparsidade de dados refere-se ao fato de que a maioria dos *tokens* do vocabulário não ocorrem em um dado texto.

A transição para modelos de linguagem neurais, iniciada por [Bengio et al. 2003], representou um avanço significativo. Ao utilizar representações vetoriais densas (*embeddings*), os modelos neurais superaram a esparsidade dos n-gramas. Arquiteturas como as Redes Neurais Recorrentes (RNNs) e suas variantes, como a LSTM, aprimoraram a capacidade de modelar o contexto sequencial. No entanto, foi a arquitetura Transformer que viabilizou a próxima grande mudança, ao permitir o processamento paralelo de sequências e a captura mais eficaz de dependências complexas.

O Transformer inaugurou a era dos modelos de linguagem pré-treinados em grande escala. Surgiram diferentes famílias de modelos: arquiteturas baseadas apenas no codificador, como o BERT [Devlin et al. 2019], especializadas em tarefas de com-

preensão de linguagem; e arquiteturas baseadas apenas no decodificador, como a série GPT [Brown et al. 2020], focadas na geração de texto. O paradigma de pré-treinamento em vastos corpora textuais, seguido por um ajuste fino para tarefas específicas, tornou-se o padrão.

O que distingue os Grandes Modelos de Linguagem (LLMs) dos seus predecessores é a escala. São modelos treinados em vastos volumes de dados textuais, como os corpora C4 [Raffel et al. 2020] e The Pile [Gao et al. 2020]. A pesquisa revelou a existência de leis de escala (*scaling laws*) [Kaplan et al. 2020]. Demonstrando que o aumento previsível no tamanho do modelo, na quantidade de dados e no orçamento computacional leva a uma melhoria de desempenho e ao surgimento de novas capacidades (*emergent abilities*) [Wei et al. 2022]. Essa descoberta impulsionou o desenvolvimento de modelos com centenas de bilhões de parâmetros, treinados em trilhões de *tokens*, definindo o cenário atual da área.

O desenvolvimento de um LLM moderno compreende três fases principais. A primeira é o **pré-treinamento**, no qual o modelo aprende a partir de um vasto corpus de texto de forma auto-supervisionada, geralmente por meio da previsão da próxima palavra. Esta fase, computacionalmente intensiva, estabelece a base de conhecimento geral e as capacidades linguísticas do modelo. A segunda fase é o **Ajuste Fino Supervisionado** (*Supervised Fine-Tuning*, SFT), onde o modelo é treinado em um conjunto de dados de alta qualidade, contendo exemplos de instruções e suas respectivas respostas desejadas. Isso o ensina a seguir instruções e a se comportar de maneira mais útil. Finalmente, a fase de **alinhamento** refina o comportamento do modelo para melhor corresponder às preferências humanas. Técnicas como o Aprendizado por Reforço com Feedback Humano (RLHF) [Ouyang et al. 2022] e, mais recentemente, a Otimização Direta por Preferência (DPO) [Rafailov et al. 2023], são empregadas para aumentar a utilidade e a segurança do modelo.

O ecossistema de LLMs é diverso, com modelos abertos e fechados. A família de modelos Llama [Touvron et al. 2023], por exemplo, oferece um balanço entre desempenho e eficiência computacional. A disponibilidade de modelos abertos é um fator importante para a pesquisa e o desenvolvimento de sistemas customizados.

Limitações e Desafios

Apesar de suas capacidades, os LLMs possuem limitações intrínsecas que são importantes para o contexto desta tese.

A primeira é o **conhecimento estático**. O conhecimento de um LLM é inerentemente estático, pois está "congelado" no tempo, refletindo apenas os dados nos quais foi treinado. Esse fenômeno, conhecido como *knowledge cutoff*, significa que o modelo não tem acesso a eventos ou informações que surgiram após a data de conclusão de seu

treinamento. Consequentemente, suas respostas podem se tornar desatualizadas ou factualmente incorretas quando a consulta exige conhecimento recente.

A segunda limitação é a propensão a **alucinações** [Ji et al. 2023]. Este termo descreve a geração de conteúdo que parece plausível e é apresentado com confiança, mas que é factualmente incorreto, irrelevante ou semanticamente sem sentido no contexto da pergunta. As alucinações não são falhas intencionais, mas subprodutos da natureza probabilística do modelo. Ele é otimizado para prever a sequência de *tokens* mais provável, não para aderir a uma base de fatos verificáveis.

Por fim, há a **falta de rastreabilidade**. A arquitetura de um LLM, com seus bilhões de parâmetros, opera como uma "caixa-preta", tornando extremamente difícil auditar a origem de uma informação específica. Não há um mecanismo direto para verificar qual fonte em seu vasto corpus de treinamento fundamenta uma determinada afirmação. Isso compromete a verificação e a confiabilidade de suas respostas.

Em conjunto, essas limitações – conhecimento estático, alucinações e falta de rastreabilidade – destacam a necessidade de arquiteturas que não dependam exclusivamente do conhecimento parametrizado do modelo. Para superar esses desafios, surgiram novos paradigmas, entre os quais se destaca a Geração Aumentada por Recuperação (RAG), detalhada na Seção 2.4. Essa abordagem busca ancorar as respostas do LLM em fontes de conhecimento externas e verificáveis. Isso mitiga os problemas de desatualização, fabricação de fatos e falta de fontes.

2.3.3 Embeddings em Grafos de Conhecimento (KGE)

Uma abordagem fundamental para a aplicação de métodos de aprendizado de máquina em dados estruturados é a técnica de *Knowledge Graph Embedding* (KGE). O objetivo central dos modelos KGE é mapear os componentes do grafo, entidades e relações, para um espaço vetorial contínuo de baixa dimensão, preservando a estrutura topológica e as propriedades inerentes ao grafo original [Wang et al. 2017].

Diferentemente dos *embeddings* de palavras (como Word2Vec ou BERT), que capturam a semântica textual, os modelos KGE focam na preservação da veracidade das triplas (h, r, t), onde h representa a entidade cabeça (*head*), r a relação e t a entidade cauda (*tail*).

Dentre as diversas famílias de modelos KGE, destacam-se os **Modelos Translacionais**, que interpretam as relações como operações de translação no espaço vetorial.

TransE (Translating Embeddings)

Proposto por [Bordes et al. 2013], o TransE é o modelo seminal desta família. Sua premissa básica é que, se uma tripla (h, r, t) é válida, a representação vetorial da

entidade cauda **t** deve estar próxima da soma da representação da entidade cabeça **h** com o vetor da relação **r**. Formalmente, espera-se que:

$$\mathbf{h} + \mathbf{r} \approx \mathbf{t} \tag{2-1}$$

O modelo define uma função de pontuação (*score function*) $f(h, r, t) = -||\mathbf{h} + \mathbf{r} - \mathbf{t}||$, que deve ser minimizada para triplas verdadeiras e maximizada para triplas falsas. Apesar de sua eficiência e simplicidade, o TransE apresenta limitações na modelagem de relações complexas, como relações 1-para-N, N-para-1 e reflexivas.

Evoluções do Modelo Translacional (TransH, TransR)

Para superar as limitações do TransE, diversas variações foram propostas:

- **TransH** [Wang et al. 2014]: Introduz a ideia de projetar as entidades em um hiperplano específico da relação antes de aplicar a translação, permitindo que uma mesma entidade tenha representações distintas, dependendo da relação em que está envolvida.
- **TransR** [Lin et al. 2015]: Avança esse conceito ao modelar entidades e relações em espaços semânticos distintos, utilizando matrizes de projeção para mapear entidades do espaço de entidades para o espaço da relação correspondente.

Implicações para a Recuperação

No contexto de sistemas de QA e RAG, técnicas de KGE podem ser utilizadas como mecanismos de recuperação densa e estrutural. Ao calcular a similaridade vetorial ou a plausibilidade de triplas no espaço latente, é possível realizar a predição de links (*link prediction*) para inferir fatos ausentes ou ranquear entidades candidatas, baseando-se puramente na estrutura do grafo, independentemente do conteúdo textual.

Contudo, modelos KGE tradicionais são inerentemente *transdutivos*, exigindo o re-treinamento para lidar com novas entidades não vistas durante o treinamento, o que os diferencia das abordagens indutivas baseadas em GNNs ou em *embeddings* textuais, adotadas nesta tese para lidar com a natureza dinâmica e aberta das perguntas em linguagem natural.

2.3.4 Redes Neurais em Grafos (GNNs)

Problemas modeláveis em grafos possuem características de difícil análise por técnicas de aprendizagem de máquina convencionais, mesmo que profundas. Logo, estudos sobre redes neurais para grafos e suas aplicações começaram a surgir a partir de 2005, com a publicação de uma proposta por Scarselli e Gori (2005) [Scarselli et al. 2005].

Em sequência, tal proposta foi generalizada no artigo *"The graph neural network"* [Scarselli et al. 2008], com boa repercussão na área. Neste trabalho, foram apresentados alguns fundamentos matemáticos das redes neurais para grafos contemporâneas. Com o avanço das arquiteturas de aprendizagem profunda, duas abordagens começaram a se destacar também nas arquiteturas de redes neurais para grafos: a recursiva e a convolucional.

As Redes Neurais para Grafos são comumente aplicadas, mas não restritas, a três atividades diferentes, sendo elas [Zhou et al. 2020]:

- **Predição de Nós**: tarefa que consiste em prever um valor ou classe de um nó em um ou múltiplos grafos. Por exemplo, prever a categoria de um artigo em uma rede de citações. Na grande maioria das vezes, essa tarefa consiste em um aprendizado semi-supervisionado em que as classes de alguns nós são informadas e a GNN é utilizada para classificar os outros nós de acordo com seus *embeddings* e ligações dentro do grafo.
- **Predição de Grafos**: nessa tarefa, o objetivo é prever um valor ou uma classe para um grafo inteiro. É muito utilizada em problemas que possuem vários grafos pequenos que precisam ser organizados em grupos. Um exemplo de aplicação é a predição de toxicidade de moléculas não conhecidas.
- **Geração de Grafos**: dado um conjunto de dados com vários grafos agrupados por similaridade, novos grafos podem ser gerados. Uma aplicação é a geração de novas moléculas de acordo com princípios ativos e a facilidade de síntese no desenvolvimento de novos medicamentos.
- **Predição de Arestas**: essa tarefa consiste na predição de um valor ou classe para uma aresta (uma ligação entre dois nós) ou, até mesmo, a existência ou não de determinada aresta. Um exemplo de aplicação é a identificação do fluxo de movimentos em uma sequência de imagens ou vídeos.

No contexto desta tese, a GNN executa uma tarefa análoga à **predição de grafos**. Seu objetivo não é classificar o grafo, mas sim aprender uma representação vetorial única e densa (*embedding*) para todo o subgrafo de conhecimento recuperado. Esse vetor agregado, que encapsula a semântica e a estrutura das evidências interconectadas, é, então, fornecido ao LLM como um *token* contextual, orientando a geração de uma resposta fundamentada na estrutura do conhecimento.

Redes Neurais Recorrentes para Grafos

As primeiras abordagens de aplicação de redes neurais em grafos foram derivadas das redes neurais recorrentes (RNNs), baseadas no trabalho de David Rumelhart [Rumelhart, Hinton e Williams 1986]. Neste tipo de rede neural, o neurônio artificial é retroalimentado com sua saída. Tal recorrência possibilita a análise de contexto (relação

da entrada atual com entradas anteriores) e foi muito utilizada no processamento de linguagem natural, entre outras atividades.

As Redes Neurais Recorrentes para Grafos combinam os conceitos das RNNs com fundamentos das cadeiras de Markov [Gagniuc 2017]. Assim, a recorrência é realizada para cada nó do grafo, tendo como entradas em cada iteração características provindas dos nós vizinhos. Logo, a medida que as recorrências são realizadas (treino da rede neural), as representações de cada nó é impactada pela sua vizinhança.

O primeiro passo dessa abordagem é a definição de representações iniciais para cada nó ou vértice do grafo. As representações costumam ser vetores gerados a partir da combinação dos atributos dos vértices. A Figura 2.3 ilustra um exemplo de grafo com representações *one-hot-encoding*, em que o vértice, por si só, é representado por uma posição no vetor de características. Assim, todas as outras posições são zeradas.

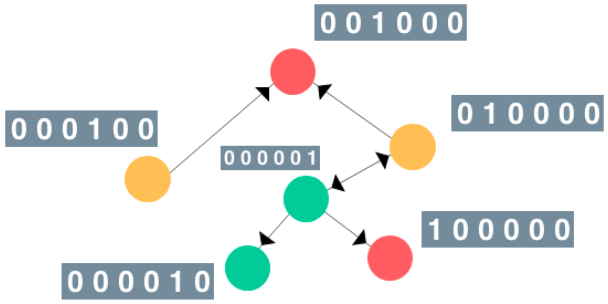

Figura 2.3: Representações (embeddings) dos vértices usando *one-hot-encoding*[Anand 2020].

Nota-se que, em aplicações reais, *one-hot-encoding* só é utilizado para geração das representações dos nós quando os mesmos não possuem atributos.

Uma vez que cada vértice possui sua representação vetorial, o próximo passo é transformar cada vértice em unidades recorrentes interligadas através de redes neurais *feed-forward* convencionais (como *multilayer perceptron*, por exemplo). A Figura 2.4 apresenta as unidades recorrentes (vértices), as *feed-forward* (arestas), e as representações (*embeddings*) para cada vértice.

Para garantir a influência da vizinhança da representação de cada nó, durante o treinamento, são enviadas mensagens pelas arestas que posteriormente serão agregadas à representação inicial do nó. Tal método foi detalhado na definição da GGNN e chamado de *Message Passing* [Li et al. 2015].

Conceito *Message Passing*

O processo de passagem de mensagem (*Message Passing*) é também chamado de agregação de vizinhança (*Neighbourhood Aggregation*). Ele consiste em enviar men-

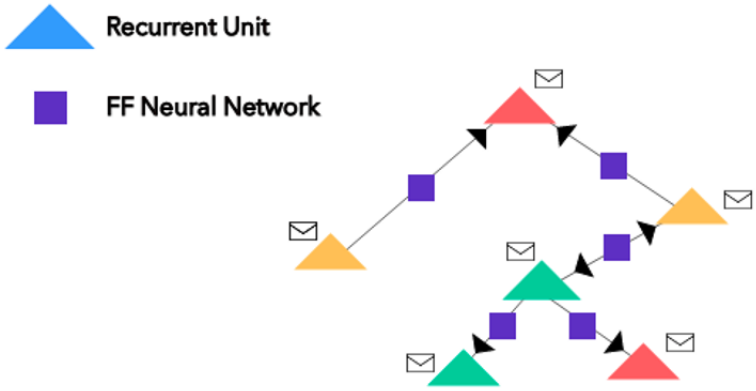

Figura 2.4: Esquema de um modelo de rede neural recorrente para grafos. Os envelopes são as representações vetoriais para cada vértice (que agora são unidades recorrentes)[Anand 2020].

sagens (os *embeddings*) dos vértices vizinhos para um dado vértice através das arestas que os interligam. Caso as arestas sejam sem direção, as mensagens são enviadas nos dois sentidos (como se existissem duas arestas direcionais em direções opostas). As redes neurais *feed-forward* são responsáveis por essa tarefa e por transformar tais mensagens de acordo com a evolução do treinamento. Elas ainda podem utilizar características da própria aresta se aplicável. Nesse caso, diferentes tipos de arestas podem ser atribuídas a diferentes redes neurais *feed-forward* a fim de realizar um processamento diferente para cada tipo de aresta.

Por exemplo, seja G um grafo que descreve interações entre usuários em uma rede social. A interação (aresta) "Seguir" entre dois usuários u_1 e u_2 pode ser mapeada por uma rede neural *feed-forward* específica. Já a interação "Curtir" entre um usuário u_3 e uma postagem p_1 pode ser mapeada por outra rede neural *feed-forward*.

Após o envio das mensagens pelas arestas, a representação é então atualizada pela unidade recorrente. Ou seja, o vetor que representa o vértice em questão é atualizado de acordo com a função recorrente definida pela rede neural recorrente. Na GGNN, essa unidade funciona exatamente como uma unidade convencional do LSTM [Hochreiter e Schmidhuber 1997] e pode ou não considerar as entradas e o contexto atual durante a geração de um novo *embedding*.

O processo descrito acima é apresentado na Figura 2.5 e realizado para todos os vértices em paralelo. Com isso, a cada iteração, a representação de um dado vértice sofre influência de uma vizinhança cada vez maior até que seja atingido o limite de vizinhança (hyperparâmetro) ou até que o nó sofra influencia de todo o grafo.

Os *embeddings* gerados podem, então, ser utilizados isoladamente para classificação de vértices, em conjunto para identificação de padrões em arestas e sub-grafos, ou agregados em um único *embedding* que representa o grafo, nesse caso, para classificação

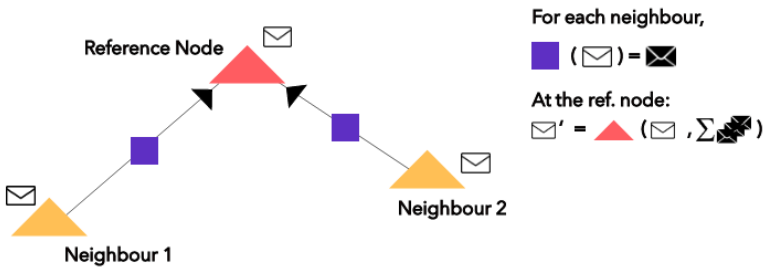

Figura 2.5: As representações vetoriais são atualizadas com base nas mensagens recebidas[Anand 2020].

do grafo, por exemplo.

Redes Neurais Convolucionais para Grafos

Como o próprio nome já diz, a proposta desse tipo de arquitetura é realizar convoluções em estruturas de dados representadas através de grafos. A primeira proposta de rede neural convolucional para grafos foi realizada em 2016 [Kipf e Welling 2016]. Os autores tomaram como base as Redes Neurais Convolucionais e utilizaram filtros espectrais localizados nas convoluções.

De forma sucinta, uma rede neural convolucional é composta por camadas convolucionais e de *pooling*. As camadas convolucionais são responsáveis pela aplicação de filtros nas entradas e extraem informações importantes de cada parte da entrada. As camadas de *pooling* pegam os dados extraídos e realizam reduções para reter apenas a informação mais importante. Ao final desse processo, os dados são enviados para uma camada totalmente conectada que realiza a atividade supervisionada (classificação, por exemplo). O processo de aprendizagem consiste na escolha dos melhores filtros para extração das informações que melhor tratam o problema em questão [Kipf e Welling 2016].

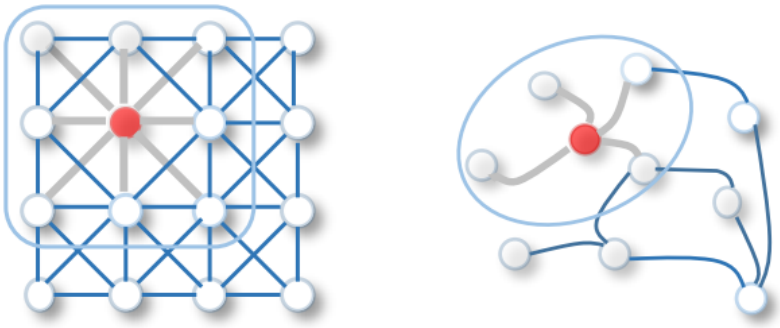

Figura 2.6: À esquerda uma representação de convolução em um dado euclidiano de duas dimensões (imagem, por exemplo). À direita, a representação da convolução em um grafo [Wu et al. 2020].

Como é possível observar na Figura 2.6, as convoluções em redes neurais convolucionais convencionais utilizam uma abordagem espacial, em um domínio de duas dimensões. Em imagens, por exemplo, pode-se entender cada pixel como um nó do grafo e a vizinhança definida pelo tamanho do filtro. Na Figura 2.6 a convolução 2D leva a média ponderada dos valores de pixel do nó vermelho junto com seus vizinhos.

Na segunda parte da Figura 2.6 é representada a convolução no grafo. Para obter uma representação oculta do nó vermelho, uma solução simples da operação convolucional do grafo é tomar o valor médio das características do nó vermelho junto com seus vizinhos. Diferente dos dados de imagem, os vizinhos do nó são não ordenados e de tamanho variável [Wu et al. 2020].

Assim como as redes neurais recorrentes para grafos, as redes neurais convolucionais para grafos também trabalham com sinais entre os nós. De fato, as convoluções são realizadas nos sinais enviados de um nó para o outro (*embeddings*). O propósito das convoluções é a aplicação de filtros para a extração das conexões (arestas) mais relevantes para representação de um dado nó. Assim, como nas redes neurais convolucionais convencionais, também existem camadas de *pooling* cujo objetivo é reduzir a complexidade de partes do grafo, extraindo apenas a informação relevante para um dado problema [Wu et al. 2020].

Graph Attention Network (GAT)

Uma limitação das arquiteturas convolucionais para grafos, como a GCN, é que elas atribuem o mesmo peso a todos os nós vizinhos durante o processo de agregação. No entanto, a importância de cada vizinho pode variar significativamente. Para endereçar essa questão, foi proposta a Rede de Atenção em Grafos (*Graph Attention Network*, GAT) [Veličković et al. 2018]. Essa arquitetura introduz um mecanismo de atenção, inspirado no sucesso dos Transformers, para ponderar a importância de diferentes nós na vizinhança.

O princípio central da GAT é calcular dinamicamente os pesos de atenção que cada nó exerce sobre seus vizinhos. Para um dado nó, o modelo computa coeficientes de atenção para cada um de seus vizinhos, indicando a relevância da informação de um para o outro. O cálculo é realizado por uma pequena rede neural *feed-forward* compartilhada entre todos os nós. Em seguida, os coeficientes são normalizados através da função *softmax*, resultando em uma distribuição de probabilidade sobre a vizinhança. A nova representação do nó é, então, uma combinação linear das representações de seus vizinhos, ponderada pelos pesos de atenção calculados.

Para estabilizar o processo de aprendizado e capturar um conjunto mais rico de características, a GAT emprega o mecanismo de atenção multi-cabeça (*multi-head attention*). Múltiplos mecanismos de atenção independentes, ou "cabeças", calculam represen-

tações em paralelo. As representações resultantes de cada cabeça são subsequentemente concatenadas ou agregadas para formar a representação final do nó. Essa abordagem permite que o modelo capture diferentes tipos de relações e aspectos da vizinhança simultaneamente. Alcançando resultados de estado da arte em diversas tarefas de aprendizado em grafos.

Graph Attention Network v2 (GATv2)

A arquitetura GAT original, embora eficaz, apresenta uma limitação teórica importante. O mecanismo de atenção é estático, ou seja, os pesos de atenção são determinados apenas pelas características dos nós. Isso significa que a função de atenção não pode ser modificada dinamicamente durante o processo de agregação. Para superar essa limitação, foi proposta a *Graph Attention Network v2* (GATv2) [Brody, Alon e Yahav 2022].

A principal inovação da GATv2 reside na reformulação do mecanismo de atenção. Enquanto a GAT original calcula os coeficientes de atenção através de uma função linear aplicada à concatenação das características dos nós, a GATv2 introduz uma função de atenção mais expressiva. O novo mecanismo permite que diferentes camadas da rede tenham diferentes pesos de atenção para os mesmos pares de nós. Essa flexibilidade adicional resulta em maior capacidade de expressão e melhor desempenho em tarefas complexas.

A GATv2 mantém a eficiência computacional da GAT original, preservando a complexidade linear em relação ao número de arestas. Simultaneamente, oferece maior poder representacional através do mecanismo de atenção dinâmica. Essa combinação de eficiência e expressividade torna a GATv2 particularmente adequada para aplicações que requerem processamento de grafos complexos com estruturas relacionais diversificadas.

2.3.5 Implicações para a Arquitetura Proposta

As arquiteturas de redes neurais são componentes centrais na arquitetura neurosimbólica proposta. Elas atuam em duas frentes distintas, mas complementares. A primeira é o motor de geração de linguagem natural e a segunda é a interface de interpretação semântica e topológica dos dados estruturados. A integração dessas duas capacidades é o que permite ao sistema conectar o conhecimento simbólico explícito com a fluidez da geração textual.

Os Grandes Modelos de Linguagem (LLMs), baseados na arquitetura Transformer, são empregados como o componente gerador final. Contudo, a arquitetura proposta não os utiliza de forma isolada. Reconhecendo suas limitações intrínsecas, como o conhecimento estático e a propensão a alucinações, o sistema é projetado para “ancorar” o LLM. Ele é condicionado a um contexto de alta qualidade, que é previamente recuperado,

estruturado e logicamente refinado, garantindo que a resposta final seja fundamentada em evidências verificáveis.

As Redes Neurais em Grafos (GNNs), especificamente a arquitetura GAT, desempenham um papel fundamental como ponte entre o domínio simbólico e o neural. Após a recuperação e a orquestração lógica do subgrafo de evidências, a GNN é responsável por processar essa estrutura e gerar uma representação vetorial densa (o *graph token*). Esse vetor encapsula a semântica e a topologia das evidências, atuando como um *soft prompt* que orienta o LLM a raciocinar sobre o conhecimento estruturado de forma mais eficaz.

A articulação conceitual desses componentes neurais na arquitetura é detalhada no Capítulo 4. As decisões de implementação, incluindo a escolha dos modelos de linguagem e a arquitetura que integra a GNN com o LLM, são aprofundadas no Capítulo 5 (Seção 5.4.5).

2.4 Geração Aumentada por Recuperação (RAG)

Os Modelos de Linguagem de Grande Escala (LLMs), apesar de sua notável capacidade de gerar texto coerente e relevante, operam com base em conhecimento paramétrico estático, que é consolidado durante a fase de treinamento.

Como dito na Seção 2.3.2, essa característica intrínseca impõe três limitações fundamentais: a incapacidade de acessar informações posteriores ao seu treinamento, a propensão a gerar "alucinações", e a falta de rastreabilidade. Para superar tais desafios, foi proposto o paradigma de Geração Aumentada por Recuperação (RAG), uma abordagem que integra um mecanismo de recuperação de informação a um modelo gerador [Gao et al. 2023].

A arquitetura RAG visa a ancorar as respostas do LLM em um corpo de conhecimento externo e dinâmico. Em vez de depender exclusivamente de seu conhecimento interno, o modelo primeiro recupera documentos ou passagens de texto relevantes para uma dada consulta. Em seguida, utiliza essas informações como contexto para formular uma resposta mais precisa, verificável e atualizada. Essa sinergia entre recuperação e geração tem se mostrado eficaz para mitigar alucinações e adaptar LLMs a domínios de conhecimento específicos sem a necessidade de re-treinamento custoso [Lewis et al. 2020].

Nesta seção, serão apresentados os fundamentos do paradigma RAG. Inicia-se com a apresentação formal do conceito e sua motivação. Em seguida, é detalhada sua arquitetura canônica, composta pelos módulos de indexação, recuperação e geração. Posteriormente, discutem-se os desafios inerentes ao RAG convencional e, por fim, introduz-se sua evolução para o uso com Grafos de Conhecimento, que estabelece a base para as contribuições desta tese.

2.4.1 O Paradigma RAG

O paradigma de Geração Aumentada por Recuperação foi introduzido por [Lewis et al. 2020] como uma estratégia para mitigar as limitações de conhecimento dos LLMs. A abordagem parte da premissa de que o conhecimento paramétrico de um modelo, por mais vasto que seja, é inerentemente estático e propenso à obsolescência. Além disso, a dependência exclusiva desse conhecimento interno é uma das causas primárias para a geração de alucinações, nas quais o modelo produz informações plausíveis, porém factualmente incorretas.

O RAG aborda diretamente essas questões ao fundamentar (*grounding*) o processo de geração de texto em fontes de conhecimento externas e explícitas. Ao condicionar a resposta não apenas à pergunta do usuário, mas também a um conjunto de evidências textuais recuperadas em tempo real, o RAG força o modelo a basear suas asserções em fatos verificáveis. Esse mecanismo aumenta a confiabilidade e a acurácia das respostas, permitindo que o conhecimento utilizado pelo sistema seja continuamente atualizado sem a necessidade de retreinar o LLM subjacente.

É importante distinguir a abordagem RAG de outras técnicas de adaptação de LLMs, como o ajuste fino (*fine-tuning*). O ajuste fino é um processo que especializa o comportamento do modelo ou internaliza conhecimento adicional em seus parâmetros. Esse processo é computacionalmente intensivo e não resolve a questão do conhecimento estático. Uma vez treinado, o conhecimento do modelo ajustado permanece congelado. O RAG, por outro lado, externaliza a base de conhecimento, permitindo atualizações dinâmicas e de baixo custo, bastando para isso modificar a fonte de dados externa.

Dessa forma, o RAG pode ser entendido como uma evolução da engenharia de prompts e do aprendizado no contexto (*in-context learning*) [Gao et al. 2023]. Enquanto a engenharia de prompts tradicional depende da inserção manual de exemplos e contexto na consulta, o RAG automatiza e otimiza esse processo. Ele transforma a tarefa de encontrar o contexto relevante em um problema de recuperação de informação. Isso permite que o sistema encontre e injete dinamicamente as informações mais relevantes de um vasto corpo de dados.

O que supera as limitações de tamanho e complexidade de um prompt construído manualmente.

Os benefícios centrais do paradigma RAG podem ser sintetizados em três áreas principais [Gao et al. 2023]:

- **Aumento da Acurácia:** Ao basear as respostas em evidências recuperadas, o RAG reduz significativamente a ocorrência de alucinações, produzindo informações factualmente mais precisas.

- **Conhecimento Atualizado:** A arquitetura permite que o sistema utilize informações em tempo real. A base de conhecimento pode ser continuamente atualizada, garantindo que as respostas do LLM não se tornem obsoletas.
- **Rastreabilidade e Verificabilidade:** Como o modelo utiliza fontes explícitas, é possível rastrear a origem da informação contida na resposta, permitindo que os usuários verifiquem a veracidade e o contexto dos dados, o que representa um avanço em relação à natureza de "caixa-preta"dos LLMs tradicionais.

A implementação eficaz de um sistema RAG, contudo, depende de uma etapa preliminar de preparação dos dados, a indexação, que estrutura a base de conhecimento para que a recuperação seja rápida e relevante.

2.4.2 Arquitetura Canônica: Indexação, Recuperação e Geração

A arquitetura de um sistema RAG canônico é modular e geralmente compreende três estágios principais: indexação, recuperação e geração, conforme ilustrado na Figura 2.7. Embora a conceituação original de [Lewis et al. 2020] se concentre nos componentes de recuperação e geração, a etapa de indexação é um pré-requisito funcional implícito para a eficiência do processo.

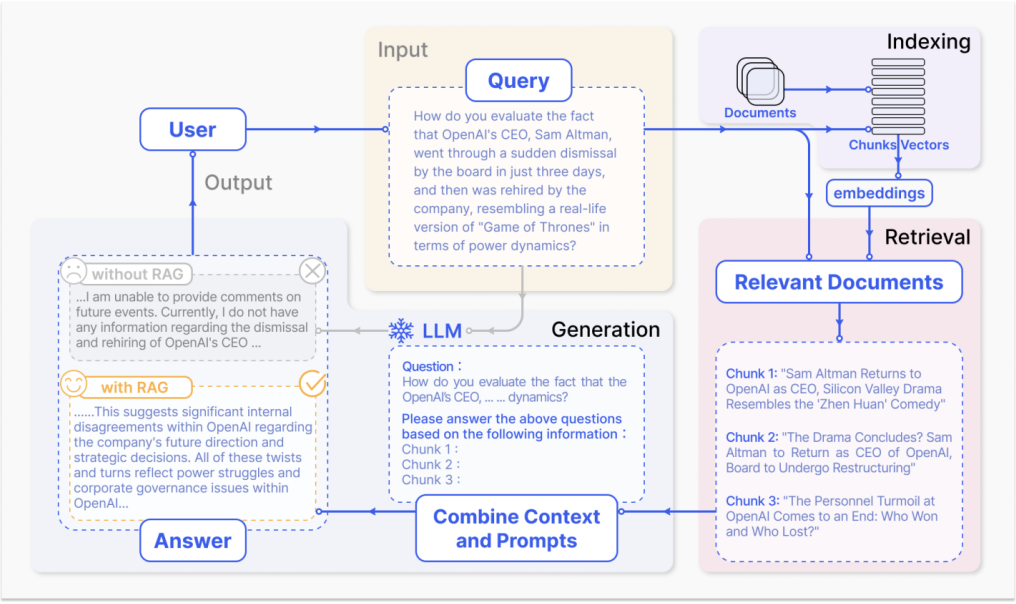

Figura 2.7: Arquitetura canônica de um sistema de Geração Aumentada por Recuperação (RAG), ilustrando os estágios de indexação, recuperação e geração. Fonte: [Gao et al. 2023].

O primeiro estágio, a **indexação**, consiste em preparar a base de conhecimento externa. Documentos brutos são processados e divididos em segmentos menores, ou "trechos"(*chunks*). Essa segmentação é uma decisão de projeto importante, pois o tamanho

dos trechos impacta diretamente a qualidade da recuperação. Em seguida, cada trecho é convertido em uma representação numérica, um vetor, por meio de um modelo de *embedding*. Esses vetores, que capturam a semântica do texto, são armazenados em um índice especializado, tipicamente um banco de dados vetorial, que permite buscas por similaridade em larga escala e com baixa latência.

O segundo estágio é o do **Recuperador** (*Retriever*). Dada uma consulta do usuário, o recuperador é responsável por buscar no índice os trechos de informação mais relevantes. A consulta do usuário é primeiramente convertida em um vetor com o mesmo modelo de *embedding* utilizado na indexação. Em seguida, o sistema realiza uma busca por similaridade (por exemplo, por similaridade de cosseno) entre o vetor da consulta e os vetores dos trechos no índice. Como resultado, o recuperador devolve um conjunto ordenado dos k trechos mais relevantes, que servirão como contexto para o próximo estágio. As estratégias de recuperação podem variar, incluindo as abordagens esparsas, densas ou híbridas discutidas na Seção 2.2.

O estágio final é o do **Gerador** (*Generator*), que é um Modelo de Linguagem de Grande Escala. Ele recebe a consulta original do usuário e o conjunto de trechos relevantes recuperados na etapa anterior. Essas informações são combinadas em um *prompt* estruturado, que instrui o LLM a formular uma resposta com base estrita no contexto fornecido. Ao condicionar a geração a essas evidências explícitas, o gerador sintetiza uma resposta final que é fundamentada, factual e alinhada com os dados da base de conhecimento externa. Isso faz com que o modelo não dependa apenas de seu conhecimento paramétrico interno. Essa síntese é o que efetivamente "aumenta"a capacidade do modelo, resultando em uma resposta mais confiável e precisa.

2.4.3 Desafios do RAG Convencional

Apesar de sua eficácia em mitigar as limitações dos LLMs, a abordagem RAG convencional não é isenta de desafios. A qualidade da resposta final é altamente dependente da qualidade de cada um dos estágios do processo. Otimizar a interação entre o recuperador e o gerador é uma tarefa complexa. Os desafios podem ser agrupados em três áreas principais: recuperação ruidosa, fidelidade da geração e contexto desestruturado [Gao et al. 2023].

O primeiro desafio reside na **recuperação ruidosa**. A relevância do contexto fornecido ao gerador é determinante para a qualidade da resposta. No entanto, o recuperador pode falhar em encontrar os documentos mais pertinentes. De maneira inversa, ele também pode recuperar um excesso de informações de forma que os trechos relevantes se perdem em meio a dados irrelevantes. Este último problema é exacerbado pelo fenômeno conhecido como “perdido no meio” (*lost in the middle*), no qual os LLMs tendem a dar

mais atenção às informações presentes no início e no fim do contexto, ignorando dados importantes que aparecem no meio [Liu et al. 2023].

O segundo desafio está relacionado à **fidelidade da geração**. Mesmo quando o contexto recuperado é perfeitamente relevante e preciso, não há garantia de que o gerador irá utilizá-lo de forma fiel. O LLM ainda pode apresentar comportamento alucinatório, ignorando, contradizendo ou extrapolando as informações fornecidas no contexto. Garantir que a resposta gerada seja estritamente fundamentada nas evidências recuperadas é um problema em aberto. Esse problema é conhecido como o desafio da atribuição ou da fidelidade (*faithfulness*) [Ji et al. 2023].

Por fim, um desafio particularmente importante para esta tese é o do **contexto desestruturado**. O RAG convencional trata a base de conhecimento como uma coleção de trechos de texto independentes. Essa abordagem, embora escalável, ignora as ricas relações semânticas e estruturais que existem entre as informações [Han et al. 2025]. Ao recuperar fatias de texto isoladas, o sistema perde a visão do todo, sendo incapaz de capturar nexos causais, hierarquias ou outras conexões complexas que não estão explicitamente contidas em um único trecho. Essa limitação inerente torna o RAG convencional inadequado para tarefas de raciocínio complexo que exigem a síntese de informações a partir de múltiplas fontes inter-relacionadas, uma lacuna que os Grafos de Conhecimento são particularmente aptos a preencher.

2.4.4 RAG em Grafos de Conhecimento (GraphRAG)

Para superar o desafio do contexto desestruturado, uma evolução natural do paradigma RAG consiste em substituir a base de conhecimento textual por uma estrutura mais rica e explícita: o Grafo de Conhecimento (como apresentado na Seção 2.1). Essa abordagem, genericamente denominada RAG em Grafos de Conhecimento (GraphRAG), modifica fundamentalmente a natureza da etapa de recuperação. Em vez de apenas buscar trechos de texto semanticamente similares à consulta, o objetivo passa a ser a identificação e extração de um subgrafo relevante que contenha as entidades e as relações necessárias para responder à pergunta [Han et al. 2025].

A ideia central do GraphRAG é que a estrutura do grafo preserva as conexões entre as informações, que são perdidas em uma coleção de documentos textuais. Ao recuperar um subgrafo, o sistema fornece ao LLM um contexto que não é apenas um conjunto de fatos, mas também um mapa de como esses fatos se relacionam entre si. Isso permite que o modelo realize um raciocínio mais complexo, seguindo caminhos e inferindo relações que não estariam aparentes em trechos de texto isolados.

O processo canônico de um sistema GraphRAG normalmente envolve duas etapas principais na fase de recuperação. Primeiramente, são identificados nós-âncora (ou

sementes) no grafo. Esses nós são as entidades mais relevantes para a pergunta do usuário, geralmente encontradas por meio de busca densa ou reconhecimento de entidades. Em seguida, a partir desses nós-âncora, o sistema expande a busca para construir um subgrafo coeso, selecionando vizinhos e caminhos que conectam as entidades relevantes. Este subgrafo resultante é então serializado para um formato textual (como uma lista de triplas) e passado ao LLM como contexto para a geração da resposta final [Han et al. 2025].

Esta seção apenas introduz o conceito de GraphRAG como o estado da arte sobre o qual esta tese se baseia. Uma análise aprofundada das diferentes estratégias e dos trabalhos relacionados será conduzida no Capítulo 3.

2.4.5 Implicações para a Arquitetura Proposta

O paradigma de Geração Aumentada por Recuperação (RAG) representa o modelo operacional central da arquitetura neuro-simbólica proposta nesta tese. O framework ARANDU é, em sua essência, uma instanciação avançada do pipeline RAG, projetado especificamente para superar os desafios do RAG convencional. A análise desses desafios, recuperação ruidosa, fidelidade da geração e o contexto desestruturado, motiva diretamente as decisões de projeto da arquitetura.

A transição do RAG convencional para o GraphRAG é o passo fundamental que alinha esta fundamentação teórica com as contribuições da tese. Ao adotar o Grafo de Conhecimento como a base de dados externa, a arquitetura proposta aborda a limitação do contexto desestruturado de forma direta, permitindo um raciocínio mais profundo sobre as evidências recuperadas.

Embora o GraphRAG ofereça uma estrutura mais rica para a recuperação de informações, ele não está isento de limitações. Uma das principais restrições é a complexidade inerente à manipulação de grandes grafos. Essa complexidade pode levar a desafios de escalabilidade e eficiência computacional. Além disso, a simples recuperação de subgrafos relevantes não garante que o LLM consiga interpretar e utilizar essas informações de forma eficaz.

É nesse contexto que a representação com Redes Neurais em Grafos (GNN) e a orquestração lógica desempenham um papel importante. As GNNs são capazes de transformar a estrutura do grafo em representações vetoriais densas, facilitando a integração com o LLM e permitindo um raciocínio mais profundo sobre as relações entre as entidades [Han et al. 2025]. A orquestração lógica, por sua vez, assegura que as inferências realizadas sejam consistentes e alinhadas com o conhecimento simbólico, mitigando o risco de alucinações e melhorando a fidelidade das respostas geradas.

A materialização desses conceitos na arquitetura proposta é detalhada no Capítulo 4, que descreve como o pipeline do ARANDU refina cada estágio do RAG. As de-

cisões de implementação, que detalham como cada componente do pipeline é construído, são aprofundadas no Capítulo 5.

2.5 Raciocínio Simbólico em Grafos

O campo da Inteligência Artificial (IA) tem sido historicamente marcado pela coexistência de duas abordagens paradigmáticas: a conexionista e a simbólica [Russell e Norvig 2020]. A abordagem conexionista, que fundamenta as redes neurais modernas, representa o conhecimento de forma distribuída e implícita em grandes conjuntos de parâmetros numéricos, aprendidos a partir de dados. Em contrapartida, a IA simbólica, muitas vezes referida como *Good Old-Fashioned AI* (GOFAI) [Haugeland 1985], opera sobre a manipulação de símbolos explícitos e regras lógicas formais, permitindo um raciocínio transparente e verificável.

Os Grafos de Conhecimento (GCs), como discutidos na Seção 2.1, são a materialização moderna de uma representação de conhecimento ideal para a IA simbólica. Sua estrutura, baseada em entidades (símbolos) e relações (predicados lógicos), fornece um substrato formal sobre o qual operações de raciocínio podem ser executadas. O objetivo do raciocínio simbólico em GCs é transcender os fatos explicitamente armazenados. Utilizando regras para inferir conhecimento novo e latente, pode tornar a base de conhecimento mais completa e poderosa. A partir de uma base de conhecimento, o raciocínio simbólico permite inferir novas informações, responder a perguntas e tomar decisões com base em fatos explícitos e regras lógicas [Russell e Norvig 2020].

O raciocínio em GCs se manifesta principalmente de duas formas complementares: indução e dedução [Hogan et al. 2021]. O **raciocínio indutivo** parte de instâncias específicas para inferir regras gerais. No contexto de GCs, isso se traduz na mineração de regras: o processo de analisar a estrutura do grafo para descobrir padrões lógicos recorrentes (e.g., inferir que se uma pessoa nasceu em uma cidade e essa cidade está em um país, então essa pessoa tem a nacionalidade daquele país). Por outro lado, o **raciocínio dedutivo** parte de regras gerais para chegar a conclusões específicas.

Uma vez que uma regra é estabelecida, a dedução a aplica aos fatos existentes no grafo para materializar novas triplas ou responder a consultas que exigem inferência (e.g., dada a regra anterior e os fatos “Marie Curie nasceu em Varsóvia” e “Varsóvia está localizada na Polônia”, pode-se deduzir o novo fato de que “Marie Curie tem nacionalidade polonesa”).

Nesta seção, serão explorados os fundamentos desses dois modos de raciocínio. Inicia-se com a discussão sobre a mineração de regras lógicas (o processo indutivo) e, em seguida, será abordado a inferência e a consulta em grafos (o processo dedutivo).

2.5.1 Mineração de Regras Lógicas

A mineração de regras lógicas é o processo indutivo de descobrir padrões estruturais recorrentes em um Grafo de Conhecimento. O objetivo é extrair um conjunto de regras de inferência que tornem explícito o conhecimento latente nos dados. Essas regras geralmente seguem a forma de **regras de Horn**, uma classe de cláusulas lógicas que são particularmente adequadas para o raciocínio computacional. Uma regra de Horn é uma implicação da forma:

$$B_1 \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow H$$

Onde H (a cabeça) é um único átomo (ou tripla), e $B_1, \ldots, B_n$ (o corpo) é uma conjunção de átomos. Cada átomo representa uma tripla do grafo, como nasceu_em($?a, ?b$), onde $?a$ e $?b$ são variáveis.

A descoberta dessas regras em GCs de grande escala é uma tarefa computacionalmente complexa. Ela envolve a busca em um vasto espaço de possíveis combinações de triplas. Sistemas especializados, como o **AMIE3** [Lajus, Galárraga e Suchanek 2020], foram desenvolvidos para realizar essa tarefa de forma eficiente. Tais sistemas exploram o grafo, propondo regras candidatas e avaliando-as com base em métricas estatísticas para determinar sua validade e relevância.

A qualidade de uma regra minerada é tipicamente avaliada por duas métricas principais: suporte e confiança. O **suporte** de uma regra é o número de vezes que o corpo da regra é verdadeiro nos dados, indicando sua generalidade. A **confiança padrão**, por sua vez, é a proporção de vezes que a cabeça da regra é verdadeira, dado que o corpo é verdadeiro.

Contudo, a confiança padrão pode ser enganosa em GCs, que operam sob a **Hipótese de Mundo Aberto** (*Open World Assumption*). A ausência de um fato não implica que ele seja falso. Para lidar com isso, métricas mais robustas como a **Confiança PCA** (*Partial Completeness Assumption*) foram propostas. A Confiança PCA refina o cálculo da confiança, estimando a probabilidade de a cabeça da regra ser verdadeira ao considerar a funcionalidade das relações no corpo da regra, oferecendo uma medida mais fidedigna da qualidade da inferência [Lajus, Galárraga e Suchanek 2020].

2.5.2 Inferência e Consulta em Grafos de Conhecimento

Após a descoberta de regras lógicas, o passo seguinte é o raciocínio dedutivo. Este processo consiste em aplicar as regras gerais aos fatos existentes para derivar conclusões específicas. A inferência enriquece o grafo ao tornar explícito o conhecimento que estava apenas implícito. Isso permite responder a consultas mais complexas, que dependem de múltiplos saltos de raciocínio.

A aplicação de regras pode ocorrer em dois momentos distintos: materialização ou em tempo de execução. A **materialização** é um processo offline. Todas as regras são aplicadas exaustivamente sobre o grafo, e as triplas inferidas são salvas. Isso acelera as consultas, pois as respostas já estão pré-calculadas. Contudo, pode levar a uma explosão no tamanho do grafo e dificulta sua atualização.

A alternativa é a **inferência em tempo de execução**. As regras são aplicadas dinamicamente no momento da consulta. Isso evita a explosão de dados e mantém o grafo conciso. Esta abordagem é mais flexível para bases de conhecimento que mudam com frequência. No entanto, o custo computacional da inferência é pago a cada consulta, o que pode aumentar a latência.

A execução da inferência e da consulta de fatos é realizada por meio de linguagens de consulta especializadas. **SPARQL** é a linguagem padrão para grafos RDF, como discutido na Seção 2.1.3. Ela utiliza padrões de triplas para consultar o grafo. Sua cláusula `CONSTRUCT` permite que ela atue como um motor de inferência, aplicando regras para gerar novas triplas que não existiam explicitamente no grafo.

Código 2.3 Exemplo de uso da cláusula CONSTRUCT em SPARQL

```
PREFIX ex: <http://example.org/>

CONSTRUCT {
  ?person ex:hasColleague ?colleague .
}
WHERE {
  ?person ex:worksAt ?company .
  ?colleague ex:worksAt ?company .
  FILTER(?person != ?colleague)
}
```

O exemplo do Código 2.3 demonstra como a cláusula `CONSTRUCT` pode ser utilizada para inferir novas relações em um grafo RDF. No exemplo, a consulta busca pares de indivíduos que trabalham na mesma empresa e infere que eles são colegas, criando uma nova tripla `ex:hasColleague` para cada par identificado.

Para Grafos de Propriedade, a linguagem proeminente é o **Cypher**, como abordado na Seção 2.1.3. Sua sintaxe utiliza "*ASCII-art*" para representar padrões de nós e relacionamentos de forma intuitiva. Embora não seja formalmente uma linguagem de inferência, sua capacidade de descrever e encontrar caminhos complexos permite a execução de raciocínio multi-salto. Uma aplicação de propósito análogo ao da inferência em muitas aplicações práticas.

2.5.3 Raciocínio Baseado em Ontologias

Enquanto a mineração de regras adota uma abordagem *bottom-up* para descobrir conhecimento, o raciocínio também pode operar de forma *top-down*. Nesse paradigma, o conhecimento é guiado por uma ontologia. Uma **ontologia** é uma especificação formal e explícita de uma conceitualização compartilhada [Gruber 1993]. Ela define o vocabulário de um domínio, incluindo classes, propriedades e seus relacionamentos.

A ontologia funciona como o esquema do grafo, ou TBox (*Terminological Box*). Ela contém os axiomas e restrições lógicas do domínio. Os fatos individuais, por sua vez, constituem o ABox (*Assertional Box*). O raciocínio dedutivo, neste contexto, utiliza um motor de inferência (*reasoner*) para aplicar os axiomas do TBox sobre os fatos do ABox, a fim de verificar a consistência dos dados e derivar conhecimento novo.

A linguagem padrão para expressar ontologias na Web Semântica é a **OWL** (*Web Ontology Language*) [Group 2012]. O OWL é, na verdade, uma família de linguagens com diferentes níveis de expressividade. Sua semântica formal é baseada em **Lógicas Descritivas** (*Description Logics* - DLs) [Baader et al. 2003]. As DLs são um subconjunto decidível da lógica de primeira ordem, otimizadas para o raciocínio sobre taxonomias e classificações.

Por meio de axiomas, é possível definir a estrutura semântica de um domínio. Por exemplo, a conhecida ontologia FOAF (*Friend of a Friend*) [Brickley e Miller 2014] define a classe `foaf:Person` como uma subclasse de `foaf:Agent`. Um *reasoner*, ao processar essa ontologia, pode usar o axioma `rdfs:subClassOf` para realizar inferências. Se o grafo contém o fato de que `(:MarieCurie rdf:type foaf:Person)`, o *reasoner* pode deduzir a tripla `(:MarieCurie rdf:type foaf:Agent)`, mesmo que esta não esteja explicitamente declarada nos dados.

2.5.4 Implicações para a Arquitetura Proposta

O arcabouço teórico do raciocínio simbólico fundamenta as decisões de projeto mais distintivas da arquitetura neuro-simbólica proposta. Enquanto os sistemas de recuperação se concentram em encontrar fatos, a integração de um módulo de raciocínio lógico permite que a arquitetura valide, enriqueça e refine ativamente as evidências encontradas.

A dualidade entre os processos indutivo e dedutivo inspira diretamente a estratégia híbrida offline/online da arquitetura. O processo indutivo de mineração de regras, por ser computacionalmente intensivo, é alocado para uma fase de preparação offline. Isso permite a construção de um banco de regras robusto sem impactar a latência da consulta. Em contrapartida, o processo dedutivo de inferência é aplicado de forma direcionada e em tempo de execução, garantindo que o raciocínio seja eficiente e estritamente relevante ao contexto da pergunta.

Adicionalmente, o uso de linguagens de consulta formais e a possibilidade de integração com ontologias fornecem o mecanismo para garantir a fidelidade e a consistência lógica. Essa capacidade de manipulação explícita do conhecimento é o que habilita o componente de refinamento, uma das contribuições centrais desta tese. O raciocínio simbólico, portanto, atua como a ponte que conecta as evidências recuperadas com a geração final, assegurando que o contexto do LLM seja logicamente sólido.

A forma como estes princípios são estruturados no fluxo de dados da arquitetura ARANDU é o foco do Capítulo 4. As escolhas tecnológicas e os detalhes da implementação de cada componente de raciocínio são aprofundados no Capítulo 5.

CAPÍTULO 3

Trabalhos Relacionados e Estado da Arte

Uma vez estabelecidos os conceitos fundamentais no capítulo anterior, este capítulo se dedica a uma análise aprofundada da literatura sobre sistemas de Pergunta e Resposta (QA) baseados em Grafos de Conhecimento. O objetivo é contextualizar esta pesquisa no panorama científico, examinando as abordagens para os desafios de recuperação e raciocínio sobre estruturas de conhecimento. A análise vai além da descrição de técnicas, focando em como as soluções foram aplicadas e suas limitações.

Para alcançar tal objetivo, o capítulo está estruturado em quatro seções. A primeira investiga a evolução dos modelos de Geração Aumentada por Recuperação (RAG). A segunda aprofunda-se nas estratégias de recuperação de informação dos grafos. A terceira examina as estratégias de raciocínio para inferência de novas informações. Por fim, a quarta seção sintetiza o conhecimento levantado, identificando as lacunas que motivam as contribuições desta tese.

3.1 Evolução do RAG Baseado em Grafos

O paradigma de Geração Aumentada por Recuperação (RAG) consolidou-se como uma arquitetura fundamental para mitigar as limitações de conhecimento dos Grandes Modelos de Linguagem (LLMs) [Gao et al. 2023]. Contudo, o campo não é estático; ele tem evoluído continuamente, buscando otimizar a qualidade e a relevância das evidências fornecidas ao componente gerador. Esta seção traça a trajetória evolutiva, partindo da sua formulação inicial, que operava sobre coleções de texto não estruturado, até as abordagens mais sofisticadas que exploram o conhecimento explícito e interconectado dos Grafos de Conhecimento. O objetivo é demonstrar como a complexidade crescente das soluções reflete uma compreensão cada vez mais profunda dos desafios inerentes à recuperação e ao raciocínio. Ao contextualizar essa evolução, a lacuna de pesquisa que esta tese se propõe a preencher é evidenciada.

3.1.1 O Paradigma: RAG Convencional sobre Texto

O paradigma RAG foi formalmente introduzido por [Lewis et al. 2020] como uma arquitetura de propósito geral para tarefas de PLN intensivas em conhecimento. A proposta original unia um recuperador de passagens de texto (retriever), baseado em representações densas, a um modelo gerador do tipo sequência-a-sequência. Esta arquitetura, hoje categorizada como RAG "vanilla" ou "ingênuo" (*Naive RAG*) [Gao et al. 2023], estabeleceu o fluxo canônico de recuperar-ler (*retrieve-read*). Nele, evidências textuais são buscadas em um corpus externo e inseridas no contexto do LLM para a síntese da resposta.

A principal limitação dessa abordagem, para o escopo desta tese, reside no tratamento do conhecimento como um **contexto não estruturado** [Han et al. 2025]. Ao operar sobre trechos de texto isolados (*chunks*), o RAG convencional ignora as ricas relações semânticas e estruturais que conectam as informações. Esta fragmentação do conhecimento impede a realização de raciocínios complexos que dependem da travessia de múltiplos fatos interligados. Este desafio motivou a transição para fontes de conhecimento estruturado.

3.1.2 A Transição para Fontes Estruturadas: Precursores do Graph-RAG

A ideia de alavancar a estrutura explícita dos GCs para aprimorar modelos de linguagem não é nova. Sua integração formal no paradigma RAG se deu por meio de trabalhos precursores que provaram a viabilidade de unir LLMs e Redes Neurais em Grafos (GNNs). Um dos trabalhos mais influentes nesse sentido foi o **QA-GNN** [Yasunaga et al. 2022]. Os autores propuseram uma arquitetura que, a partir de entidades em uma pergunta, recupera um subgrafo relevante do GC e o processa com uma GNN. As representações dos nós, agora enriquecidas com o contexto estrutural, são então utilizadas para guiar um modelo de linguagem (BERT) a identificar a entidade-resposta correta. O QA-GNN estabeleceu um precedente fundamental: o de que a representação neural da topologia do grafo é benéfica para o raciocínio do modelo de linguagem. O Módulo de Representação Neural via GNN do ARANDU se baseia nesse princípio.

Outras abordagens, como o **GreaseLM** [Zhang et al. 2022], exploraram um acoplamento ainda mais forte, fundindo as representações textuais e de grafo em múltiplas camadas intermediárias. Esses trabalhos demonstraram que a sinergia neuro-estrutural era uma fronteira promissora, pavimentando o caminho para a formalização do GraphRAG.

3.1.3 A Formalização do Raciocínio em Grafos no Contexto RAG

Com a viabilidade técnica estabelecida, a pesquisa evoluiu da simples utilização do grafo para a aplicação de mecanismos de raciocínio mais explícitos e fiéis sobre sua estrutura. O trabalho **Reasoning on Graphs (RoG)** [Luo et al. 2024] foi um dos primeiros a focar na fidelidade (*faithfulness*) da geração do LLM. A metodologia proposta extrai caminhos de raciocínio explícitos do grafo, que são então usados para guiar e verificar a resposta gerada, garantindo que ela seja consistente com a estrutura do conhecimento.

Simultaneamente, o **Think-on-Graph (ToG)** [Sun et al. 2024] propôs uma abordagem de raciocínio iterativo. Em vez de um único passo de recuperação, o ToG permite que o LLM interaja ativamente com o grafo, explorando suas conexões para refinar progressivamente as etapas de seu raciocínio e chegar a uma conclusão mais robusta. Essas abordagens deslocaram o foco da simples recuperação de fatos para a construção de cadeias de raciocínio válidas sobre o grafo.

3.1.4 A Consolidação do Paradigma GraphRAG e Suas Variações

O ano de 2024 marcou a consolidação do "GraphRAG" como um campo de pesquisa robusto, com o surgimento de múltiplas arquiteturas de alto impacto, cada uma com focos distintos. A iniciativa **GraphRAG da Microsoft** [Edge et al. 2024] focou na escalabilidade e aplicabilidade em domínios complexos, utilizando técnicas como a detecção de comunidades para criar resumos semânticos do grafo.

Paralelamente, o **G-Retriever** [He et al. 2024] se estabeleceu como o estado da arte na otimização da recuperação e da representação neural. Suas contribuições centrais foram duas: (1) a modelagem da extração de subgrafos como um problema de otimização formal, resolvido com o Prize-Collecting Steiner Tree (PCST); e (2) o uso de uma GNN para codificar o subgrafo em um *soft prompt*, resolvendo o problema da perda de informação na linearização textual.

Finalmente, trabalhos como o **RuleRAG** [Chen et al. 2024] começaram a reintroduzir o componente simbólico de forma explícita, utilizando regras lógicas para guiar e refinar o processo de recuperação. Esta vertente reconhece que a conectividade estrutural, por si só, não é suficiente, e que a consistência lógica é importante para a fidelidade do raciocínio.

Ao analisar essas frentes, percebe-se que, embora avançadas em seus respectivos nichos, elas operam de forma relativamente isolada. O G-Retriever foca na otimização neural da recuperação, enquanto o RuleRAG foca na orientação baseada em regras lógicas (ainda que deixando o raciocínio final ao LLM). A lacuna que emerge, e que esta tese se propõe a preencher, é a ausência de uma arquitetura que integre sinergicamente a otimização da recuperação estrutural com uma camada explícita de orquestração lógica.

Garantindo que o contexto entregue ao LLM seja não apenas estruturalmente coeso, mas também logicamente completo e priorizado.

3.2 Análise de Estratégias de Recuperação em Grafos

A recuperação de informação em Grafos de Conhecimento apresenta desafios distintos. Diferentemente de fontes textuais, a relevância da evidência em um grafo depende de dois fatores. O primeiro é o conteúdo semântico dos nós e arestas. O segundo é a estrutura topológica que os interconectam. A eficácia de um sistema de recuperação reside na sua capacidade de lidar com essa dualidade de forma integrada e eficiente.

Diante dessa complexidade, a literatura científica propõe diferentes famílias de estratégias. Cada uma delas aborda o problema da recuperação sob uma ótica particular. Esta seção se dedica a uma análise técnica e comparativa dessas abordagens. Elas serão categorizadas em três grupos principais, detalhados nas subseções a seguir.

A primeira categoria engloba as abordagens de indexação de grafos. Tais métodos focam em pré-processar a estrutura para facilitar uma busca inicial. A segunda categoria discute as abordagens focadas em caminhos, que se concentram em extrair sequências de raciocínio entre entidades. A terceira categoria analisa as abordagens de otimização de subgrafos, que buscam encontrar um subconjunto de evidências que equilibre relevância e coesão. A compreensão dessas estratégias fundamenta as decisões de projeto adotadas nos mecanismos de recuperação do framework ARANDU.

3.2.1 Abordagens de Indexação de Grafos

A etapa de indexação em um sistema GraphRAG visa criar uma representação pesquisável de uma base de conhecimento estruturada. O objetivo é permitir a identificação eficiente de pontos de entrada relevantes no grafo a partir de uma consulta em linguagem natural. As estratégias para alcançar esse objetivo variam em complexidade. Desde a indexação de componentes individuais até a criação de sumários hierárquicos da estrutura do grafo.

A abordagem mais fundamental consiste em tratar cada nó e aresta do grafo como um documento independente. Nessa estratégia, são construídos dois tipos de índices paralelos. Um índice lexical, baseado em algoritmos como o BM25 (como visto na Seção 2.2), é criado sobre as propriedades textuais dos componentes (e.g., nomes e descrições). Simultaneamente, um índice vetorial armazena os *embeddings* desses mesmos textos, permitindo a busca semântica. Esta abordagem, adotada como base pelo ARANDU, viabiliza uma primeira camada de recuperação híbrida, mas não indexa a topologia do grafo de forma explícita.

Estratégias mais avançadas buscam indexar a própria estrutura do grafo. A abordagem **GraphRAG** proposta por [Edge et al. 2024] utiliza algoritmos de detecção de comunidades para particionar o grafo em agrupamentos de nós densamente conectados. Em seguida, um LLM é empregado para gerar sumários textuais para cada uma dessas comunidades. O resultado é um índice hierárquico que permite à busca navegar de resumos de alto nível para as regiões mais relevantes do grafo, uma técnica eficaz para exploração em grafos de larga escala.

Uma perspectiva alternativa é apresentada por trabalhos como o **Knowledge Graph Prompting (KGP)** [Wang et al. 2024]. Em vez de indexar um grafo pré-existente, esta abordagem constrói um grafo que funciona como um índice sobre múltiplos documentos. Nessa estrutura, os nós representam unidades textuais, como parágrafos ou tabelas, e as arestas indicam relações de similaridade semântica ou de adjacência estrutural. Tal método é particularmente útil para tarefas de QA multi-documento, onde as relações entre as fontes de informação são parte essencial do contexto.

3.2.2 Abordagens Focadas em Caminhos

Uma segunda família de estratégias de recuperação aborda o problema a partir da perspectiva do raciocínio multi-salto. Em vez de focar na indexação global do grafo, as abordagens focadas em caminhos partem de um conjunto de nós-âncora relevantes e se concentram em identificar as sequências de relações mais significativas que os conectam. O objetivo é extrair cadeias de raciocínio explícitas, que podem ser mais facilmente interpretadas pelo LLM do que um subgrafo denso e não ordenado.

O trabalho **PathRAG** [Chen et al. 2025] exemplifica esta abordagem. A metodologia proposta busca ativamente por caminhos relacionais entre as entidades identificadas na pergunta. O sistema primeiro identifica os caminhos mais promissores e, em seguida, utiliza essas estruturas lineares para podar (*prune*) o espaço de busca, recuperando apenas os nós e relações que fazem parte desses caminhos. Essa estratégia reduz a quantidade de informação irrelevante (ruído) passada ao LLM, focando o contexto nas evidências que compõem uma linha de inferência direta.

De forma análoga, a metodologia **Reasoning on Graphs (RoG)** [Luo et al. 2024] também utiliza a extração de caminhos como mecanismo central para garantir a fidelidade da geração. O sistema extrai múltiplos caminhos de raciocínio do grafo que conectam as entidades da pergunta à resposta. Esses caminhos são então apresentados ao LLM como a evidência fundamental sobre a qual a resposta deve ser gerada. O foco do RoG está em usar os caminhos como uma forma de prova ou justificação para a resposta final.

Outras abordagens, como a do **Think-on-Graph (ToG)** [Sun et al. 2024], utili-

zam a travessia de caminhos como parte de um processo de raciocínio iterativo, onde o LLM explora ativamente o grafo. Embora eficazes para tarefas que demandam um raciocínio linear, as abordagens focadas em caminhos podem ter dificuldade em capturar contextos onde múltiplas linhas de evidência convergem de forma não sequencial. Um cenário melhor tratado por abordagens de otimização de subgrafos.

3.2.3 Abordagens de Otimização de Subgrafos

Uma terceira vertente de pesquisa em recuperação de informação em grafos vai além da extração de caminhos lineares. As abordagens de otimização de subgrafos buscam extrair um subconjunto de nós e arestas que, em conjunto, formam um contexto coeso, relevante e informacionalmente denso. O objetivo é encontrar um balanço ótimo entre a inclusão de evidências pertinentes e a exclusão de informações ruidosas, tratando a recuperação como um problema de otimização formal.

Uma estratégia basilar para a extração de subgrafos é a expansão de vizinhança `k-hop`. A partir de um conjunto de nós-âncora, o método recupera todos os nós e arestas a uma distância de até `k` saltos. Embora simples, essa abordagem é heurística e frequentemente ineficaz, pois recupera uma grande quantidade de informação irrelevante. Ela não dispõe de um mecanismo para ponderar a importância relativa das diferentes conexões, podendo poluir o contexto do LLM.

Para superar essa limitação, o **G-Retriever** [He et al. 2024] introduziu uma abordagem que formaliza a recuperação como uma instância do problema da Árvore de Steiner Coletora de Prêmios (*Prize-Collecting Steiner Tree*, PCST). Nessa formulação, os nós e arestas do grafo recebem "prêmios" (*prizes*) proporcionais à sua relevância semântica para a pergunta do usuário. As arestas, por sua vez, possuem "custos" (*costs*) associados à sua inclusão no subgrafo resultante [He et al. 2024].

O algoritmo PCST busca, então, por um subgrafo em formato de árvore que maximize a soma dos prêmios dos nós coletados, subtraída da soma dos custos das arestas utilizadas. O resultado é um subgrafo conectado que, por princípio, contém as evidências mais relevantes (altos prêmios), conectadas da forma mais concisa possível (baixo custo). Essa metodologia oferece uma solução formal para o equilíbrio entre relevância e ruído.

Uma abordagem alternativa de otimização é apresentada pelo modelo **SURGE** [Kang et al. 2023]. Em vez de resolver um problema de otimização global para encontrar uma única árvore de conexão, o SURGE adota uma estratégia de ranqueamento local. O processo ocorre em dois estágios: primeiro, um recuperador de granularidade grossa identifica um conjunto inicial de entidades relevantes. Em seguida, uma GNN atua como um ranqueador de granularidade fina, que aprende a pontuar e selecionar os melhores subgrafos de um salto (*one-hop*) ao redor dessas entidades candidatas. A otimização,

neste caso, consiste em aprender a classificar e escolher as vizinhanças mais informativas, sendo uma técnica eficaz para diálogos baseados em conhecimento.

A análise dessas abordagens avançadas demonstra a importância da otimização na recuperação. Tanto a otimização global via PCST quanto o ranqueamento local de subgrafos representam o estado da arte para a recuperação estruturada em sistemas GraphRAG.

Dada a sua robustez teórica e eficácia empírica, a otimização via PCST foi adotada na implementação de referência do framework ARANDU, conforme detalhado no Capítulo 5.

3.3 Análise de Estratégias de Raciocínio em RAG

Após a recuperação de um subgrafo de evidências, a etapa subsequente em um pipeline RAG é a de raciocínio. Este processo envolve a interpretação e a síntese das informações recuperadas para formular uma resposta. A forma como este raciocínio é conduzido constitui um diferenciador central entre as arquiteturas propostas na literatura. O local onde a inferência ocorre varia, de mecanismos implícitos no modelo neural a sistemas simbólicos explícitos.

Esta seção analisa as principais estratégias de raciocínio documentadas. As abordagens são categorizadas em três paradigmas principais. O primeiro consiste na delegação do raciocínio às capacidades emergentes dos LLMs. O segundo envolve a integração com sistemas de raciocínio formais, ou *solvers*. O terceiro utiliza regras explícitas para guiar a contextualização do LLM. A análise comparativa dessas estratégias contextualiza as decisões de projeto da arquitetura do ARANDU.

3.3.1 Raciocínio Delegado ao LLM

A estratégia mais prevalente em sistemas RAG avançados consiste em delegar a tarefa de raciocínio inteiramente ao LLM [Gao et al. 2023]. Nessa abordagem, o esforço principal da arquitetura se concentra na etapa de recuperação. O objetivo é construir um contexto de evidências o mais rico, preciso e bem-estruturado possível. A premissa subjacente é que um LLM com capacidade suficiente, ao ser condicionado a um contexto de alta qualidade, pode executar implicitamente os passos de inferência necessários para formular a resposta correta [He et al. 2024].

O **G-Retriever** [He et al. 2024] é um exemplo representativo deste paradigma. A arquitetura emprega técnicas sofisticadas de otimização (PCST) e de representação neural (GNNs) para gerar um *soft prompt* que encapsula a topologia do subgrafo de evidências. Embora o G-Retriever utilize GNNs para representação estrutural, essa codificação neu-

ral serve primariamente para preparação do contexto, não constituindo um sistema de raciocínio simbólico explícito. Após essa elaborada preparação do contexto, a tarefa de interpretar as relações estruturais e sintetizar a resposta final é completamente delegada às capacidades internas do LLM. Não há uma camada de validação ou raciocínio simbólico explícito após a recuperação.

Uma variação mais interativa desta mesma filosofia é apresentada pelo **Think-on-Graph (ToG)** [Sun et al. 2024]. Diferente de receber um contexto estático, o ToG permite que o LLM explore ativamente o Grafo de Conhecimento. O modelo gera passos de raciocínio intermediários, que por sua vez disparam novas operações de busca no grafo, criando um processo iterativo. Apesar dessa interatividade, a lógica que guia a exploração e a derivação de conclusões a cada passo ainda é gerada pelo próprio LLM.

A delegação do raciocínio ao LLM oferece alta flexibilidade, aproveitando a capacidade de generalização e compreensão de linguagem natural desses modelos. No entanto, esta abordagem apresenta limitações significativas. A principal delas é a falta de transparência, uma vez que o processo de raciocínio ocorre de forma implícita nos parâmetros do modelo, operando como uma "caixa-preta" [Russell e Norvig 2020]. Adicionalmente, não há garantias de fidelidade; o LLM pode interpretar erroneamente o contexto, ignorar evidências ou cometer erros lógicos, resultando em alucinações, mesmo quando as informações corretas estão presentes [Ji et al. 2023]. Essas deficiências motivam a investigação de estratégias de raciocínio mais explícitas e verificáveis.

3.3.2 Integração com Solvers Formais

Uma abordagem alternativa ao raciocínio delegado consiste na separação explícita entre a compreensão da linguagem e a execução da lógica. Neste paradigma, o LLM atua como uma interface de linguagem natural para um motor de raciocínio simbólico, ou *solver* formal. O papel do modelo neural é traduzir a pergunta do usuário em uma representação formal e executável. O raciocínio propriamente dito é então externalizado para um sistema determinístico, como um motor de banco de dados ou um provador de teoremas.

O mecanismo central desta abordagem é a análise semântica (*semantic parsing*). Trata-se do processo de mapear uma sentença em linguagem natural para uma representação de significado que seja computacionalmente interpretável [Berant et al. 2013]. Em sistemas RAG, o LLM é instruído a realizar essa tarefa, convertendo a pergunta do usuário em uma consulta em uma linguagem formal, como SPARQL ou Cypher. A execução dessa consulta sobre o Grafo de Conhecimento é o que efetivamente realiza a inferência e recupera a resposta.

O trabalho **KnowledGPT** [Wang et al. 2023] é um exemplo desta estratégia. O

sistema utiliza um LLM para gerar consultas formais que são executadas diretamente sobre uma base de conhecimento externa. A responsabilidade do LLM se encerra na tradução da intenção do usuário para uma consulta bem-formulada; o raciocínio subsequente é realizado de forma determinística pelo motor de busca do grafo.

A principal vantagem da integração com *solvers* formais é a alta precisão e verificabilidade dos resultados. Uma vez que a consulta formal é gerada corretamente, a execução é um processo lógico e transparente, cujo resultado é garantidamente consistente com os fatos da base de conhecimento. No entanto, esta abordagem apresenta uma fragilidade (*brittleness*) considerável [Plenz e Frank 2024]. O sucesso de todo o processo depende da capacidade do LLM de gerar uma consulta sintática e semanticamente correta. Um pequeno erro na tradução pode levar a uma falha na execução ou a um resultado incorreto, sem um mecanismo robusto de recuperação de erro. Esse dilema entre precisão e fragilidade motiva a busca por arquiteturas híbridas.

3.3.3 Contextualização Guiada por Regras

Entre os paradigmas de raciocínio delegado e a integração com *solvers* formais, emerge uma terceira abordagem híbrida. A contextualização guiada por regras busca equilibrar a flexibilidade dos LLMs com o rigor do raciocínio simbólico. O princípio central não é substituir o LLM, mas sim aprimorar seu desempenho, utilizando regras lógicas para refinar, validar e enriquecer o contexto de evidências antes da etapa de geração. O objetivo é fornecer ao modelo um contexto não apenas factualmente relevante, mas também logicamente coeso e completo.

O **RuleRAG** [Chen et al. 2024] é um trabalho representativo desta categoria. A arquitetura utiliza um conjunto de regras lógicas, previamente definidas ou mineradas, para governar o pipeline de recuperação. As regras podem atuar como filtros para descartar informações inconsistentes, como mecanismos para expandir o conjunto de evidências com fatos inferidos, ou para reordenar os documentos recuperados com base em sua consistência lógica. Nesse modelo, as regras funcionam como "guardrails" simbólicos que restringem o espaço de atuação do componente neural, aumentando a fidelidade do processo.

Outras abordagens, como a do **Chain of Knowledge (CoK)** [Li et al. 2023], compartilham o mesmo princípio de construir um contexto estruturado. Embora não utilize regras mineradas, o CoK recupera triplas factuais de um Grafo de Conhecimento para formar uma cadeia de raciocínio explícita. Essa cadeia é então apresentada ao LLM, garantindo que a evidência seja apresentada de forma conectada e verificável, em vez de como um conjunto de textos isolados.

Este paradigma híbrido oferece um compromisso estratégico. Ele mantém o

LLM como o motor principal para a compreensão da pergunta e para a síntese da resposta em linguagem natural, mitigando a fragilidade dos *solvers* formais. Ao mesmo tempo, a aplicação de regras explícitas sobre o contexto aumenta a transparência e a confiabilidade do raciocínio, abordando a natureza de "caixa-preta" do raciocínio puramente delegado. A arquitetura do *framework* ARANDU, proposta nesta tese, se insere e avança este paradigma. Ela implementa uma etapa de **orquestração lógica** que não apenas aplica regras para expansão, mas também descobre e pondera caminhos de inferência no subgrafo, como será detalhado nos Capítulos 4 e 5.

3.4 Síntese e Identificação da Lacuna de Pesquisa

A análise da literatura demonstra uma clara trajetória de evolução nos sistemas RAG. A área progrediu de abordagens convencionais, que operam sobre texto não estruturado, para o paradigma GraphRAG, que alavanca a riqueza topológica dos Grafos de Conhecimento. Trabalhos recentes refinaram ainda mais este paradigma, introduzindo otimizações na recuperação de subgrafos e iniciando a integração de regras lógicas. Apesar desses avanços, a análise revela a persistência de um conjunto de limitações interconectadas, que definem uma lacuna de pesquisa clara no estado da arte.

A principal limitação reside no **acoplamento fraco entre a recuperação estrutural e o raciocínio simbólico**. Arquiteturas como o G-Retriever [He et al. 2024] otimizam a extração de um subgrafo estruturalmente coeso, mas delegam o raciocínio sobre ele inteiramente ao LLM. Inversamente, abordagens como o RuleRAG [Chen et al. 2024] empregam regras lógicas, mas sem a mesma formalização na otimização da recuperação e representação do subgrafo. Falta, portanto, uma arquitetura que integre sinergicamente a otimização da recuperação estruturada com uma camada explícita de orquestração lógica, que valide, enriqueça e refine o subgrafo recuperado.

Essa lacuna impacta diretamente a **fidelidade e a precisão** das respostas. Sem um mecanismo de orquestração lógica explícito, o contexto fornecido ao LLM, mesmo que estruturado, pode ser logicamente incompleto ou inconsistente. Isso mantém o risco de o modelo gerar respostas que não são estritamente fundamentadas em uma cadeia de evidências verificável [Ji et al. 2023]. Adicionalmente, muitas abordagens dependem de múltiplos ciclos de inferência de LLMs de grande porte para realizar o raciocínio [Sun et al. 2024], o que eleva o **custo computacional** e a latência, em desacordo com a necessidade de sistemas eficientes.

Diante do exposto, esta tese se propõe a preencher a lacuna identificada por meio de uma arquitetura neuro-simbólica integrada. O framework ARANDU é projetado para unir a recuperação de subgrafos otimizada (módulo de recuperação estruturada) a uma camada de orquestração lógica explícita (módulo de orquestração lógica). Esta camada

atua *a priori*, refinando o contexto para garantir sua consistência lógica antes da geração. Ao delegar o raciocínio complexo a componentes simbólicos e a representação topológica a uma GNN especializada (módulo de representação neural via GNN), a arquitetura viabiliza o uso de LLMs compactos. Assim, o framework endereça o desafio da eficiência computacional. O Capítulo 4 detalha essa proposta arquitetural.

CAPÍTULO 4

Uma Arquitetura Neuro-Simbólica para Geração Aumentada por Recuperação Fundamentada

Após a análise dos desafios de fidelidade e eficiência inerentes aos sistemas de Resposta a Perguntas (QA) e a identificação de uma lacuna crítica nas abordagens contemporâneas, este capítulo apresenta em detalhe a arquitetura neuro-simbólica proposta para superar tais limitações. Esta arquitetura serve como o projeto conceitual para o *framework* implementado e denominado **ARANDU**, uma solução concebida sob o princípio de uma integração sinérgica entre os paradigmas neural e simbólico. Para tal, o capítulo está organizado em torno da descrição das etapas, fases e módulos conceituais da arquitetura.

4.1 Visão Geral da Arquitetura

A arquitetura proposta aborda os desafios de cobertura, precisão e eficiência em tarefas de QA sobre Grafos de Conhecimento (KGQA). A abordagem integra métodos neurais e simbólicos de forma complementar. Diferentemente de sistemas que tratam recuperação e geração como etapas isoladas, esta arquitetura implementa um fluxo de processamento multifásico. Neste fluxo, o conjunto de evidências é progressivamente recuperado, refinado e validado antes da síntese da resposta final. Este processo visa assegurar que o LLM opere sobre um contexto de qualidade elevada na etapa final. Consequentemente, foca no aumento da probabilidade de obtenção de respostas corretas e factualmente fundamentadas.

A Tabela 4.1 apresenta a organização da arquitetura proposta em etapas, fases e módulos. Como é possível observar, o funcionamento se divide em duas etapas principais: **etapa de preparação offline** e **fluxo de execução online**.

A etapa de preparação offline, apresentada na Figura 4.1, é executada uma única vez para cada fonte de dados. Ela estabelece os artefatos necessários para a realização

da recuperação de evidências e raciocínio lógico. Nela, o conjunto de dados original é abstraido em um formato padrão. Em seguida, o Grafo de Conhecimento (GC) é armazenado para posterior consulta estruturada. A partir do GC, dois módulos são executados em paralelo: (1) o módulo de indexação híbrida, no qual as partes textuais das entidades e relações são codificadas em vetores (*embeddings*). Estes vetores são armazenados em um índice vetorial para busca semântica rápida. Adicionalmente, é criado um índice léxico para busca por similaridade textual (2) o módulo de enriquecimento lógico, no qual regras lógicas são mineradas e materializadas diretamente no grafo, opcionalmente criando uma versão enriquecida que torna as inferências mais explícitas. A geração de uma versão enriquecida do grafo é opcional, pois pode ser computacionalmente custosa para grandes grafos.

Etapa	**Fase**	**Módulo**	**Objetivo Principal**
Preparação (Offline)		Módulo de Abstração do Conjunto de Dados	Carregar e abstrair o conjunto de dados em um formato padrão, criando o Grafo de Conhecimento (GC) para posterior consulta estruturada.
		Módulo de Indexação Híbrida	Criar índices vetoriais e lexicais a partir das entidades e relações do Grafo de Conhecimento (GC) para viabilizar a recuperação textual eficiente.
		Módulo de Mineração de Regras Lógicas	Minerar regras lógicas a partir da estrutura do GC para criar um Banco de Regras a ser usado para enriquecimento do grafo recuperado.
Execução (Online)	Fase I: Recuperação Híbrida de Evidências	Módulo de Recuperação Textual	Identificar um conjunto inicial de entidades e relações candidatas com base na similaridade lexical e semântica com a pergunta do usuário.
		Módulo de Recuperação Estruturada	Extrair um subgrafo coeso e informacionalmente denso a partir dos candidatos, otimizando a relação sinal-ruído.
	Fase II: Orquestração Lógica do Contexto	Módulo de Enriquecimento Lógico	Expandir o subgrafo de evidências com triplas inferidas via regras lógicas, completando caminhos para o raciocínio.
		Módulo de Ponderação de Caminhos	Identificar os caminhos mais relevantes no subgrafo e ponderar suas arestas, priorizando as evidências para a próxima fase.
	Fase III: Representação Neural e Geração da Resposta	Módulo de Representação Neural (GNN)	Codificar o subgrafo logicamente ponderado em uma representação vetorial densa (*graph token*) que preserva a topologia para o LLM.
		Módulo de Geração Textual (LLM)	Sintetizar a resposta final em linguagem natural, utilizando um LLM compacto condicionado pelo *graph token* e pelo contexto textual.

Tabela 4.1: Descrição das Etapas, Fases e Módulos da Arquitetura Proposta

A etapa de execução online, apresentada na Figura 4.2, descreve um fluxo que é ativado a cada nova consulta e opera em três fases sequenciais: (I) Recuperação Híbrida de Evidências, (II) Orquestração Lógica do Contexto, e (III) Representação Neural e Geração da Resposta.

A **Fase I - Recuperação Híbrida de Evidências** é responsável por extrair um subgrafo candidato do grafo de conhecimento global. Seu objetivo é balancear a cobertura (*recall*) e a precisão (*precision*), iniciando com uma busca semântica ampla para garantir que nenhuma informação relevante seja perdida e finalizando com uma otimização estrutural para formar um contexto coeso.

A **Fase II - Orquestração Lógica do Contexto** constitui a parte conceitualmente mais inovadora da arquitetura. Nela, as evidências recuperadas são submetidas a um processo de enriquecimento e ponderação. Primeiramente, o grafo é expandido com triplas inferidas logicamente, garantindo a completude do contexto. Em seguida, os

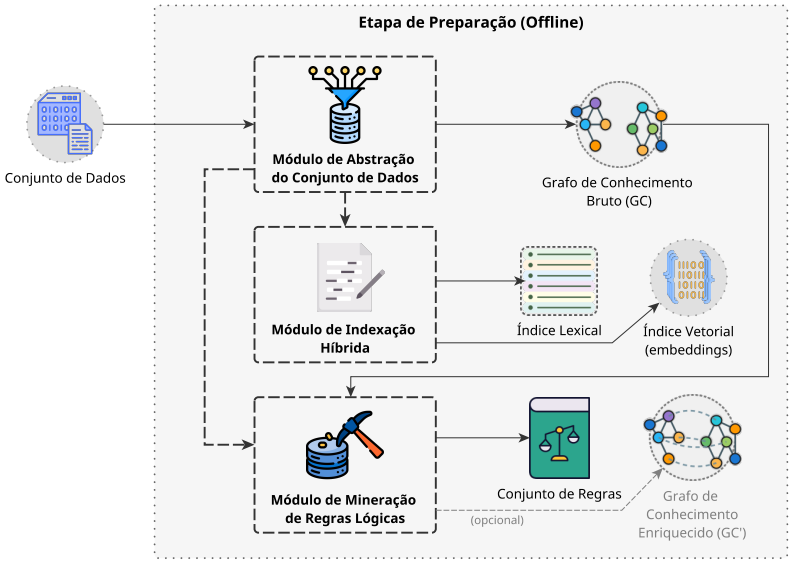

Figura 4.1: Diagrama de alto nível da arquitetura da etapa de preparação. Apresenta módulos para abstrair a fonte de dados, indexar e enriquecer a base de conhecimento.

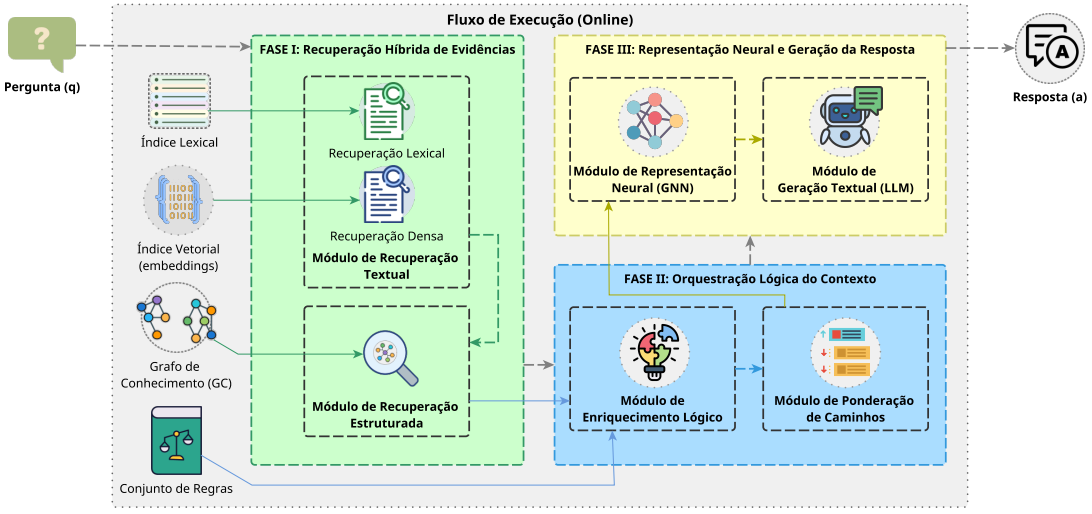

Figura 4.2: Diagrama de alto nível da arquitetura do fluxo de execução. Ilustra as três fases operacionais e o fluxo de dados desde a pergunta do usuário até a resposta final gerada pelo LLM.

caminhos do grafo são ponderados para destacar as trilhas mais relevantes em relação à pergunta, assegurando a coesão lógica do contexto que seguirá para a próxima fase.

Finalmente, a **Fase III - Representação Neural e Geração da Resposta** atua como uma ponte entre a representação simbólica do conhecimento e os modelos de linguagem neurais. O subgrafo, já enriquecido e ponderado, é codificado por uma GNN em um formato vetorial denso, compreensível pelo LLM. Este, por sua vez, utiliza esse contexto estruturado e validado para formular a resposta final em linguagem natural, concluindo o ciclo de QA.

Conforme mencionado no Capítulo 1 (Seção 1.4), essas três fases são implementadas através de seis módulos conceituais da arquitetura. A Fase I é executada pelos Módulos de **Recuperação Textual** e **Recuperação Estruturada**. A Fase II é realizada pelos Módulos de **Enriquecimento Lógico** e **Ponderação de Caminhos**. A Fase III, por sua vez, é executada pelos Módulos de **Representação Neural via GNN** e **Geração Textual**. As seções subsequentes deste capítulo se aprofundarão em cada uma dessas fases e módulos, detalhando os processos, as implementações e as justificativas teóricas por trás de cada decisão de projeto.

4.2 Etapa de Preparação e Indexação (Offline): Fundamentação do Conhecimento

A eficácia de qualquer sistema de Geração Aumentada por Recuperação (RAG) é fundamentalmente limitada pela qualidade e pela acessibilidade de sua base de conhecimento externa. Um grafo de conhecimento bruto, por mais rico que seja, muitas vezes contém conhecimento implícito e não está otimizado para a recuperação rápida de informações semânticas complexas, que é uma característica das perguntas em linguagem natural. A etapa de preparação e indexação, portanto, é um processo offline crítico que transforma a base de conhecimento bruta em um conjunto de artefatos de alta performance, prontos para serem consumidos pelo *pipeline* de execução em tempo real [Cao et al. 2025, Guo et al. 2025].

A arquitetura proposta adota uma estratégia de **preparação dual**, projetada para equipar a base de conhecimento com duas capacidades complementares: acessibilidade semântica e profundidade lógica. Por um lado, para habilitar a recuperação neural rápida e de alta similaridade semântica, o conteúdo textual associado às entidades e relações do grafo é processado e estruturado em um índice vetorial. Por outro lado, para fortalecer a capacidade de raciocínio e garantir a fidelidade, a estrutura do grafo é enriquecida com regras lógicas explícitas, materializando inferências que de outra forma permaneceriam latentes [Chen et al. 2024]. As subseções a seguir detalham cada um desses processos de preparação.

4.2.1 Fundamentação Lexical e Semântica: Indexação Híbrida

O módulo de indexação híbrida da arquitetura proposta é responsável pela preparação para recuperação textual de evidências. Como diz o nome, ele fundamenta-se em uma abordagem híbrida de indexação lexical e semântica. São construídos dois tipos de índices complementares: um índice lexical tradicional, baseado em técnicas como *inverted index* (e.g., BM25[Robertson e Zaragoza 2009]), e um índice vetorial

semântico. O índice lexical permite buscas exatas e eficientes por termos presentes nas consultas, sendo especialmente útil para perguntas que contêm nomes próprios, siglas ou termos raros que podem não ser capturados por *embeddings* semânticos. Já o índice vetorial semântico, alinhado ao estado da arte em sistemas RAG [Gao et al. 2023], projeta tanto a consulta quanto o conteúdo textual do grafo em um espaço vetorial de alta dimensão, onde a proximidade geométrica entre vetores reflete a similaridade semântica [Lewis et al. 2020]. Essa combinação visa superar a disparidade fundamental entre a linguagem natural da consulta do usuário e a estrutura formal do grafo de conhecimento, permitindo ao sistema identificar pontos de entrada relevantes no grafo a partir de perguntas abertas, tanto por similaridade exata quanto aproximada.

O processo conceitual de indexação ocorre em três passos. Primeiramente, para cada nó (entidade) e aresta (relação) no Grafo de Conhecimento, são extraídos os conteúdos textuais associados, como rótulos e descrições. Para alguns tipos de grafos de conhecimento em que não há atributos textuais além dos nomes das entidades e relações, o conteúdo textual associado pode ser o próprio nome. Também é possível realizar o enriquecimento do grafo com outras fontes de informação, como descrições de entidades e relações em documentos de referência, como, por exemplo, a Wikipedia. Os textos são, então, tratados como "documentos" individuais que representam semanticamente cada elemento do grafo.

Em segundo lugar, cada um desses documentos é processado por um modelo de linguagem pré-treinado, como um *sentence-transformer* [Reimers e Gurevych 2019], que o converte em um vetor de alta dimensão (*embedding*). Esses vetores capturam as nuances semânticas do conteúdo, permitindo uma comparação eficiente com a consulta do usuário. A escolha por esta classe de modelos se justifica por sua alta eficiência e capacidade comprovada em tarefas de busca semântica [Gao et al. 2023, Lewis et al. 2020].

Finalmente, os *embeddings* gerados são armazenados em uma estrutura de dados otimizada para a busca por vizinhos mais próximos de forma aproximada (Approximate Nearest Neighbor - ANN) [Muja e Lowe 2014], enquanto os textos originais (rótulos, descrições e nomes) são indexados em um índice lexical tradicional. O resultado desta etapa offline são dois artefatos essenciais: um índice vetorial de alta performance e um índice lexical robusto, que juntos servem como entradas para a Fase de Recuperação (Seção 4.3.1).

4.2.2 Fundamentação Lógica: Enriquecimento por Mineração de Regras

Complementar à indexação, a segunda vertente da preparação offline, que constitui o Módulo de Mineração de Regras Lógicas, foca em um desafio intrínseco aos Grafos

de Conhecimento (GCs): a sua esparsidade estrutural e incompletude lógica. Muitos GCs, embora vastos, contêm apenas um subconjunto das relações factuais possíveis, deixando muitas inferências lógicas de forma implícita. Essa esparsidade representa um obstáculo significativo para tarefas de QA multi-salto (*multi-hop*), onde a ausência de uma única tripla explícita em uma cadeia de raciocínio pode impedir a descoberta da resposta correta [Sun et al. 2024]. Para mitigar essa limitação, a arquitetura proposta inclui um módulo de **mineração de regras lógicas**, um processo que extrai padrões de inferência válidos a partir da estrutura do grafo para posterior aplicação na expansão lógica do contexto.

O processo conceitual é conduzido por algoritmos de mineração de regras lógicas, que analisam a estrutura de triplas existentes para descobrir padrões de inferência com alta confiança estatística. Um exemplo clássico é a regra de transitividade para a relação "localizado em": `(?a, localizadoEm, ?b)` $\wedge$ `(?b, localizadoEm, ?c)` $\Rightarrow$ `(?a, localizadoEm, ?c)`. A utilização de regras mineradas para extrair conhecimento de alta qualidade de GCs é uma abordagem estabelecida que pode guiar e aprimorar sistemas de raciocínio [Chen et al. 2024]. As regras descobertas são então filtradas com base em métricas de confiança para garantir sua validade e relevância.

As regras de alta confiança são aplicadas iterativamente sobre o grafo original para inferir e adicionar novas triplas que não estavam explicitamente presentes. Este processo pode ser executado ainda na etapa de preparação, gerando um **Grafo de Conhecimento Enriquecido** (G'), que é estruturalmente mais denso e logicamente mais completo que sua versão original. Bem como apenas na etapa de execução, considerando apenas o subgrafo recuperado pela Fase de Recuperação (Seção 4.3.1). Esta segunda abordagem é necessária para lidar com o caso em que o grafo de conhecimento é muito grande, e a materialização de regras na etapa de preparação é impraticável.

4.3 Fluxo de Execução (Etapa Online): Da Pergunta à Resposta Fundamentada

Com a conclusão da etapa de preparação offline, a base de conhecimento encontra-se devidamente indexada, enriquecida com as regras lógicas e pronta para ser utilizada para execução do *pipeline* de Resposta a Perguntas (QA). Esta seção dedica-se a uma exposição e detalhamento do fluxo de execução em tempo real, o *pipeline* sequencial que é acionado a cada nova consulta em linguagem natural submetida pelo usuário. O propósito central deste fluxo é promover a transformação sistemática da pergunta inicial em uma resposta final que seja não apenas correta, mas também logicamente fundamentada e explicável. Para tal, o *pipeline* é estruturado em três fases macro, cada uma delas responsável por uma camada específica de processamento e refinamento da informação.

Essas fases exploram, de maneira sinérgica, os artefatos produzidos na preparação offline. Notadamente, o Índice Vetorial de *embeddings* semânticos, o Índice Lexical, o conjunto de regras lógicas e o Grafo de Conhecimento, opcionalmente enriquecido por regras lógicas. Dessa forma, garantindo que a evidência utilizada pelo LLM seja progressivamente filtrada, aumentada e ponderada.

Como dito anteriormente, o *pipeline* é operacionalizado por três fases macroestruturais: (I) Recuperação Híbrida de Evidências, (II) Orquestração Lógica e Representação Neural, e (III) Geração da Resposta Final. Logo, o processamento inicia-se por uma etapa de recuperação ampla, focada em maximizar a cobertura das evidências. Subsequentemente, a arquitetura introduz sua principal inovação: uma fase que primeiro orquestra o subgrafo recuperado, expandindo-o com inferências e ponderando a relevância de seus caminhos, e depois o traduz em uma representação neural que preserva sua estrutura. Por fim, a geração da resposta é realizada por um LLM compacto, que se beneficia de um contexto enriquecido e logicamente priorizado. As subseções seguintes aprofundam a descrição de cada uma dessas fases, detalhando os módulos conceituais e os algoritmos que lhes dão suporte, bem como suas inter-relações no contexto da arquitetura proposta.

4.3.1 Fase I: Recuperação Híbrida de Evidências

A primeira fase do fluxo de execução online é responsável pela tarefa crítica de identificar e extrair, a partir da vasta e complexa base de conhecimento, um subgrafo de evidências que seja conciso, relevante e suficiente para responder à pergunta do usuário. A qualidade desta fase é determinante para o sucesso de todo o *pipeline*, pois uma recuperação falha ou ruidosa pode comprometer a capacidade de raciocínio do LLM na etapa final. Para enfrentar o duplo desafio de abranger a relevância semântica da pergunta e, ao mesmo tempo, respeitar a coerência estrutural do grafo, a arquitetura proposta emprega uma estratégia de **recuperação híbrida** em duas etapas, alinhada com a tendência de combinar diferentes paradigmas de busca para obter resultados mais robustos [Cao et al. 2025, Ma et al. 2025].

Um aspecto fundamental da abordagem adotada nesta fase é a **não delegação da tarefa de busca ao LLM**, em contraste com estratégias recentes que utilizam o próprio modelo de linguagem para navegar ou recuperar caminhos no grafo [Ma et al. 2025, Sun et al. 2024]. Ao optar por mecanismos dedicados de recuperação textual e otimização estrutural, a arquitetura preserva sua característica de baixo custo computacional, evitando a sobrecarga e o custo de inferência associados à execução iterativa de LLMs durante a busca. Essa decisão de design garante que a etapa de recuperação permaneça eficiente e escalável, ao mesmo tempo em que mantém a independência e a interpretabilidade dos critérios de seleção de evidências, delegando ao LLM apenas a etapa final de geração da

resposta.

A estratégia de recuperação híbrida adotada funciona como um fluxo progressivo: inicia-se com uma busca semântica inicial abrangente, capaz de capturar um conjunto amplo de entidades e relações potencialmente relevantes. Em seguida, a próxima etapa expande e aumenta esse conjunto inicial de evidências, com o objetivo explícito de elevar a cobertura (*recall*). Ou seja, maximizar as chances de incluir a entidade ou relação que efetivamente responde à pergunta, mesmo ao custo de introduzir algum ruído adicional.

Especificamente, a primeira etapa, conduzida pelo **Módulo de Recuperação Textual**, utiliza os índices vetorial e lexical para realizar uma busca semântica ampla, identificando evidências conceitualmente próximas à pergunta. Em seguida, a segunda etapa, implementada pelo **Módulo de Recuperação Estruturada**, aplica um algoritmo de otimização estrutural que expande o conjunto inicial, conectando e enriquecendo as evidências por meio de caminhos no grafo, de modo a formar um subgrafo mais coeso e informativo. Essa expansão visa garantir que, mesmo que a busca semântica inicial não recupere diretamente a resposta, ela possa ser alcançada por meio de conexões no grafo.

Módulo de Recuperação Textual

O **Módulo de Recuperação Textual**, é projetado como um módulo independente e reutilizável, responsável por realizar a busca semântica inicial sobre o Grafo de Conhecimento já indexado. Sua arquitetura conceitual é composta por três elementos principais: a interface de entrada, o mecanismo de busca vetorial e a interface de saída, todos organizados para garantir modularidade, eficiência e fácil integração com as etapas subsequentes do *pipeline*.

Entradas Conceituais A entrada principal do módulo é a **pergunta do usuário** em linguagem natural. Opcionalmente, o módulo pode receber parâmetros adicionais, como o número de evidências a serem recuperadas (k), restrições de tipo (por exemplo, buscar apenas entidades ou relações), ou filtros contextuais. Tais filtros são conceitualmente derivados de configurações ou pré-processamentos da etapa offline, garantindo que qualquer filtragem relevante já esteja definida antes do início da recuperação online.

Processo de Busca Vetorial O processo realizado pelo módulo segue os seguintes passos conceituais:

1. **Codificação da Pergunta:** A pergunta é processada por um modelo de linguagem pré-treinado (o mesmo utilizado na etapa offline), gerando um vetor denso que representa seu significado semântico.

2. **Consulta ao Índice Vetorial:** O vetor da pergunta é utilizado para consultar o índice vetorial, que já contém os *embeddings* de todas as entidades e relações do grafo. A busca retorna os k elementos mais similares, de acordo com a métrica de similaridade (tipicamente, similaridade do cosseno).
3. **Pós-processamento Opcional:** Dependendo da configuração, podem ser aplicados filtros adicionais (por exemplo, eliminar duplicatas, priorizar certos tipos de entidades, ou aplicar limiares de similaridade).

Processo de Busca Lexical (Opcional) Além da busca vetorial semântica, o módulo pode incorporar, de forma opcional e complementar, um **processo de busca lexical**. Esta etapa visa aumentar a cobertura (*recall*) inicial, especialmente em cenários onde a correspondência exata de termos ou rótulos é relevante para a tarefa de QA. A busca lexical consiste em identificar entidades e relações cujos rótulos textuais apresentam correspondência léxica (exata ou parcial) com termos presentes na pergunta do usuário.

O fluxo conceitual da busca lexical é o seguinte:

1. **Extração de Termos-Chave:** A pergunta é analisada para identificar termos ou expressões potencialmente relevantes (e.g., via *tokenização* ou extração de n-gramas).
2. **Correspondência com Rótulos do Grafo:** Cada termo-chave é comparado com os rótulos textuais das entidades e relações indexadas no grafo, utilizando critérios de igualdade, prefixo, sufixo ou similaridade lexical.
3. **Agregação dos Resultados:** Os elementos do grafo que apresentam correspondência lexical são agregados ao conjunto de evidências recuperadas, podendo receber um score específico ou serem combinados à busca semântica, conforme configuração.

A integração da busca lexical ao módulo de recuperação textual é especialmente útil para capturar casos em que a semântica da pergunta coincide diretamente com a nomenclatura do grafo, ou para lidar com perguntas factuais e nomes próprios. Além disso, a busca lexical pode ser utilizada como mecanismo de *fallback* para perguntas que não obtêm resultados satisfatórios na busca vetorial, aumentando a robustez do sistema.

Saídas Conceituais A saída do módulo são dois **conjuntos ordenados de evidências**, um para entidades e outro para relações, cada um composto por:

- Referência ao elemento do grafo (ID da entidade ou relação).
- Valor da similaridade com a pergunta (*score*).
- Metadados opcionais (tipo, rótulo textual, etc.).

Esses conjuntos são entregues à próxima etapa do *pipeline*: a recuperação estruturada.

Características Arquiteturais e Modulares Uma característica fundamental deste módulo é sua **modularidade**. Ele é concebido como um serviço desacoplado com uma interface bem definida, permitindo sua substituição ou atualização independente. Essa flexibilidade facilita a experimentação com diferentes modelos de *embeddings* ou estratégias de busca sem impactar o restante do *pipeline*. Adicionalmente, a separação clara entre entrada, processamento e saída garante **transparência e auditabilidade**, aspectos essenciais para sistemas de QA explicáveis, e assegura sua reutilização em diferentes domínios e conjuntos de dados.

Síntese do Papel no *Pipeline* O módulo de recuperação textual desempenha um papel estratégico no *pipeline* da arquitetura, funcionando como um filtro semântico de cobertura ampla, responsável por identificar, de maneira eficiente, um conjunto inicial de evidências potencialmente relevantes à pergunta do usuário. Sua concepção modular e orientada à eficiência contribui diretamente para a escalabilidade e adaptabilidade do sistema, estabelecendo uma base sólida sobre a qual as etapas subsequentes de recuperação estruturada e orquestração lógica podem operar de forma robusta e eficaz.

Módulo de Recuperação Estruturada

O **Módulo de Recuperação Estruturada**, representa o segundo estágio da fase de recuperação híbrida. Sua função é transformar o conjunto inicial de evidências semanticamente relevantes (a saida do Módulo de Recuperação Textual) em um subgrafo coeso, informativo e estruturalmente conectado. Diferentemente da recuperação textual, que privilegia a similaridade semântica, esta etapa explora as relações topológicas e a conectividade inerente ao Grafo de Conhecimento. O objetivo central é garantir que as evidências selecionadas estejam integradas em um contexto relacional que potencialize o raciocínio subsequente.

Entradas Conceituais As entradas conceituais deste módulo são: (1) um **conjunto de entidades e relações candidatas** com seus respectivos scores de relevância semântica; (2) o **Grafo de Conhecimento enriquecido**; e (3) **parâmetros de restrição**, como limites de complexidade do subgrafo.

Processo de Recuperação de Subgrafo O processo de recuperação estruturada é conceitualmente fundamentado na aplicação de algoritmos que otimizam a seleção de um subgrafo. Neste aspécto, o projeto modular da arquitetura se destaca por permitir abordagens distintas para cada cenário. Para casos nos quais cada pergunta possui um grafo correlato, pode ser adequada a utilização de algoritmos de otimização de subgrafos, como o **Prize-Collecting Steiner Tree (PCST)** [He et al. 2024], que encontra uma estrutura

conectada ótima que equilibra a relevância e a complexidade do subgrafo. Essa abordagem pode ser substituída por algoritmos de busca por caminhos (*pathfinding*) ou métodos de poda baseados em fluxo [Chen et al. 2025], em cenários com grafos de conhecimento com alta densidade, sem alterar a arquitetura fundamental.

Cumpre salientar que, embora os módulos de recuperação textual e estruturada sejam, por concepção, intercambiáveis do ponto de vista arquitetural, a interface estabelecida entre eles foi projetada com rigor metodológico para assegurar uma **integração sinérgica e acoplamento funcional elevado** entre ambas as abordagens. Tal delineamento visa garantir que a transição entre a seleção semântica inicial e a subsequente expansão estrutural se realize de maneira fluida e consistente.

Saídas Conceituais A saída do módulo é um **subgrafo de evidências estruturado**, contendo as triplas que formam a solução ótima do problema de otimização. Este subgrafo é então encaminhado para a Fase II, onde sua consistência lógica será validada.

Síntese do Papel Arquitetural O módulo de recuperação estruturada desempenha um papel fundamental ao garantir que o contexto a ser analisado nas fases seguintes não seja apenas semanticamente relevante, mas também logicamente conectado e informativamente denso. Ao transformar um conjunto disperso de candidatos em um subgrafo coeso, este módulo potencializa a capacidade de raciocínio do sistema e prepara o terreno para a orquestração lógica e o aumento neural subsequentes.

4.3.2 Fase II: Orquestração Lógica do Contexto

A Fase II, denominada **Orquestração Lógica do Contexto**, tem como objetivo central transformar o subgrafo de evidências recuperado em um contexto mais informativo e logicamente consistente. Esta etapa adiciona precisão e alinhamento com os requisitos de fidelidade exigidos para a geração de respostas por LLMs. Diferentemente de sistemas RAG convencionais, que frequentemente repassam ao LLM um contexto bruto, redundante ou inconsistente, esta arquitetura introduz uma fase intermediária de refinamento.

Esta fase é responsável por atacar de forma explícita dois desafios estruturais recorrentes em sistemas de Geração Aumentada por Recuperação: (i) a presença de ruído contextual, isto é, a inclusão de evidências irrelevantes ou distrativas que podem induzir o modelo ao erro, e (ii) a ausência de garantias lógicas sobre a consistência e a validade das evidências apresentadas ao modelo [Chen et al. 2024, Sun et al. 2024]. Ao propor uma fase dedicada ao enriquecimento e ponderação do contexto, a arquitetura busca não apenas mitigar esses desafios, mas também estabelecer um novo patamar de precisão na preparação das evidências que fundamentarão a resposta.

O pipeline desta fase, após a recuperação inicial, realiza primeiramente um enriquecimento lógico do subgrafo, adicionando arestas faltantes com base em regras previamente materializadas. Em seguida, os caminhos do subgrafo são ponderados com base na relevância contextual para a resposta. O resultado é um subgrafo enriquecido e ponderado, que servirá de entrada para a Fase III, onde será codificado por uma GNN e utilizado para a geração da resposta.

Dessa forma, a arquitetura busca aprimorar a qualidade das respostas e mitigar tanto o risco de alucinações quanto o desperdício de recursos computacionais com informações supérfluas.

Módulo de Enriquecimento Lógico

O **Módulo de Enriquecimento Lógico** é o primeiro componente da Fase II. Sua função é expandir o subgrafo de evidências recuperado, adicionando triplas inferidas a partir de regras lógicas. Este processo visa aumentar a conectividade do grafo e materializar conhecimento implícito. Tal conhecimento é essencial para um raciocínio mais completo na etapa subsequente. A abordagem utiliza regras previamente mineradas para garantir que o enriquecimento seja factualmente consistente com o domínio.

Entradas Conceituais As principais entradas deste módulo são:

- **Subgrafo de evidências recuperado**: Proveniente da etapa de recuperação estruturada, composto por triplas de entidades e relações, com seus respectivos scores de relevância e descrições textuais.
- **Conjunto de regras lógicas**: O conjunto de regras lógicas gerado na etapa de preparação (Módulo de Enriquecimento Lógico), que expressa padrões válidos no domínio do grafo.
- **Parâmetros de configuração**: Configurações da expansão lógica, como a confiança e suporte mínimos para a aplicação das regras, e a utilização de triplas sintéticas.

Processo de Enriquecimento Lógico O processo consiste na expansão lógica do subgrafo, na qual as regras são aplicadas sobre as triplas do subgrafo recuperado para gerar novas triplas. Este mecanismo pode inferir tanto triplas já existentes no grafo global quanto triplas sintéticas, que representam novo conhecimento. A utilização de triplas sintéticas é um parâmetro configurável, permitindo ajustar o balanço entre cobertura e precisão.

Saídas Conceituais A saída do módulo é um **subgrafo enriquecido**, caracterizado por:

- **Subgrafo expandido**: O conjunto de triplas original, acrescido das triplas inferidas logicamente.
- **Metadados de inferência**: Informações que rastreiam quais regras geraram cada nova tripla, para fins de explicabilidade.

Este subgrafo enriquecido serve como entrada para o Módulo de Ponderação de Caminhos.

Síntese do Papel Arquitetural O Módulo de Enriquecimento Lógico desempenha um papel importante ao transformar o subgrafo de evidências em um contexto mais denso e completo. Sua função arquitetural é materializar conhecimento implícito, inferindo novas triplas que conectam informações de maneira logicamente consistente. Ao fazer isso, o módulo prepara o grafo para um raciocínio mais robusto na etapa de ponderação, garantindo que o conhecimento latente no domínio seja explicitamente considerado.

Módulo de Ponderação de Caminhos

O **Módulo de Ponderação de Caminhos** atua sobre o subgrafo enriquecido para identificar e priorizar as evidências mais relevantes. Sua função é reduzir o ruído contextual, focando o processamento subsequente nos caminhos de inferência mais promissores. Para isso, o módulo atribui pesos às arestas que refletem sua importância relativa à pergunta do usuário.

Entradas Conceituais As principais entradas deste módulo são:

- **Subgrafo enriquecido**: Proveniente do Módulo de Enriquecimento Lógico.
- **Pergunta do usuário**: Utilizada como referência para o cálculo de relevância.
- **Parâmetros de configuração**: Definem a estratégia e os limiares para o ranqueamento.

Processo de Ponderação de Caminhos O processo é centrado no ranqueamento de caminhos relevantes. Para tal, os resultados do módulo de recuperação textual são reutilizados para identificar as entidades e relações mais relevantes para a pergunta. Os caminhos que possuem arestas ou entidades relevantes para a pergunta são pontuados de acordo com a relevância das arestas e entidades, e da quantidade de arestas e entidades relevantes. As triplas que compõem os caminhos são então ponderadas de acordo com a pontuação dos caminhos que elas compõem. Triplas com presença em multiplos caminhos pontuados recebem como pontuação a soma das pontuações dos caminhos.

Saídas Conceituais A saída do módulo é um **subgrafo refinado e ponderado**, caracterizado por:

- **Subgrafo com Arestas Ponderadas**: Um conjunto de triplas em que as arestas foram ponderadas para destacar os caminhos de inferência mais relevantes.
- **Metadados de Ranqueamento**: Informações sobre os pesos atribuídos, permitindo a rastreabilidade do processo de priorização.

Este subgrafo refinado é então encaminhado para a Fase III, para ser codificado por uma GNN e utilizado para a geração da resposta.

Síntese do Papel Arquitetural O Módulo de Ponderação de Caminhos cumpre a função estratégica de filtrar e priorizar as evidências dentro do subgrafo enriquecido. Seu papel arquitetural é reduzir o ruído contextual, destacando os caminhos de inferência mais relevantes para a pergunta do usuário. Ao atribuir pesos às triplas, este módulo assegura que a subsequente representação neural (Fase III) seja construída a partir de um contexto otimizado, aumentando a probabilidade de uma geração de resposta precisa e focada. A ponderação também é relevante para a representação textual do grafo, pois garante que as informações mais relevantes sejam apresentadas prioritariamente no contexto fornecido ao LLM.

4.3.3 Fase III: Representação Neural e Geração da Resposta

A Fase III, denominada **Representação Neural e Geração da Resposta**, representa o estágio final do *pipeline*, responsável por traduzir o contexto estruturado e logicamente validado em uma resposta precisa e fluente em linguagem natural. Esta fase inicia com a codificação do subgrafo ponderado em uma representação neural compacta e, em seguida, utiliza essa representação para condicionar um LLM a gerar a resposta final.

O processo é dividido em dois módulos sequenciais. Primeiro, o **Módulo de Representação Neural via GNN** transforma o subgrafo em um *graph token*, um vetor denso que encapsula a semântica e a estrutura das evidências. Este *token* atua como um *soft prompt* que preserva o viés indutivo relacional do grafo. Em seguida, o **Módulo de Geração Textual** utiliza este *graph token*, juntamente com a pergunta original e a representação textual das evidências, para instruir um LLM a sintetizar a resposta.

Esta abordagem de duas etapas permite que modelos de linguagem de menor porte, como modelos na ordem de 3B a 8B de parâmetros, alcancem desempenho comparável ou superior a modelos maiores. A arquitetura da Fase III foi projetada com foco na **eficiência computacional** e na **fidelidade das respostas**. Ao receber um contexto pré-processado em um formato otimizado, o LLM pode concentrar sua

capacidade de processamento na tarefa de geração textual, reduzindo a latência e os custos computacionais.

Módulo de Representação Neural via GNN

O **Módulo de Representação Neural via GNN**, constitui a etapa final da Fase II. Sua responsabilidade é transformar o subgrafo logicamente enriquecido e ponderado em uma representação vetorial contínua que preserva tanto a semântica das entidades quanto a estrutura relacional do grafo. A decisão arquitetural central deste módulo é a adoção de uma Rede Neural em Grafo (GNN), como a *Graph Attention Network v2* (GATv2) [Brody, Alon e Yahav 2022], para codificar o subgrafo em um *"graph token"* compacto, que pode ser eficientemente processado pelo LLM.

Um aspecto distintivo desta abordagem é a **preservação explícita do viés indutivo relacional** através do processamento neural especializado, em contraste com estratégias convencionais que linearizam a informação do grafo em formato textual, uma prática comum em sistemas RAG tradicionais e adotada por frameworks como GraphRAG [Han et al. 2025] ou o LightRAG [Guo et al. 2025]. Ao utilizar uma arquitetura neural específica para grafos, evita-se a perda de informações topológicas cruciais que ocorre durante a conversão para texto plano, mantendo a riqueza estrutural necessária para um raciocínio mais preciso.

A estratégia de codificação via GNN funciona como um processo de destilação informacional: parte do subgrafo refinado, já filtrado logicamente, e aplica sucessivas camadas de agregação e atenção para concentrar as informações mais relevantes em uma representação densa e informativa. Esta representação final, denominada *graph token*, atua como um **soft prompt** contextual [Lester, Al-Rfou e Constant 2021] que condiciona a geração do LLM sem necessidade de modificações na arquitetura do modelo de linguagem ou processos custosos de ajuste fino.

Entradas Conceituais As principais entradas deste módulo são:

- **Subgrafo refinado**: Conjunto de triplas logicamente válidas provenientes do módulo de orquestração lógica de evidências, com suas respectivas pontuações de relevância e metadados estruturais.
- **Embeddings pré-computados**: Representações vetoriais das entidades e relações, obtidas durante o processo offline de enriquecimento da base.
- **Parâmetros de configuração**: Dimensionalidade do *graph token* e hiperparâmetros do modelo.

Processo Conceitual de Codificação Neural O processo de transformação do subgrafo em *graph token* é realizado através das seguintes etapas sequenciais:

1. **Inicialização dos Embeddings de Nós**: Cada entidade no subgrafo refinado é inicializada com sua representação vetorial (*embedding*) obtida da base de dados vetorial, a mesma utilizada durante o processo de recuperação textual (Módulo de Recuperação Textual). Esta reutilização garante consistência semântica e otimiza o desempenho computacional.
2. **Codificação das Relações**: As arestas do subgrafo são representadas através de *embeddings* específicos de tipo de relação, permitindo que a GNN distinga entre diferentes tipos de conexões semânticas.
3. **Propagação Multi-Camada**: A GNN aplica múltiplas camadas de propagação de mensagens, onde cada nó agrega informações de seus vizinhos através de um mecanismo de atenção, capturando padrões relacionais em diferentes escalas de vizinhança.
4. **Agregação Global com Atenção**: Após a propagação local, uma operação de *pooling* baseada em atenção global é aplicada para gerar uma representação única do subgrafo completo.
5. **Projeção para o Espaço do LLM**: A representação agregada é projetada para o espaço dimensional do modelo de linguagem através de uma camada de transformação linear, garantindo compatibilidade com a arquitetura do *transformer*.

Integração com o Modelo de Linguagem O *graph token* resultante é integrado ao LLM através de uma estratégia de concatenação contextual, onde a representação do subgrafo é inserida como um token especial na sequência de entrada do modelo. Esta abordagem, inspirada em técnicas de integração multimodal [Li et al. 2023], permite que o LLM tenha acesso simultâneo à informação textual da pergunta e à representação estrutural enriquecida do contexto recuperado.

A integração é realizada de forma que o *graph token* atue como um **contexto estrutural** que complementa a informação textual, sem interferir nos mecanismos internos de atenção do LLM. Diferentemente de abordagens que requerem modificações arquiteturais no modelo de linguagem [Yasunaga et al. 2022] ou processos custosos de ajuste fino (*fine-tuning*), esta estratégia mantém a compatibilidade com modelos pré-treinados, facilitando a adoção e reduzindo significativamente os custos computacionais e de desenvolvimento.

Características Arquiteturais O módulo de representação neural via GNN é projetado com as seguintes características distintivas:

- **Modularidade**: A arquitetura permite que o módulo de codificação neural seja substituído por diferentes famílias de modelos de grafos, garantindo flexibilidade e extensibilidade.

- **Escalabilidade**: O design suporta o processamento de subgrafos de tamanhos variados, mantendo a complexidade computacional controlada, por exemplo, através de técnicas conceituais como a amostragem de vizinhança [Hamilton, Ying e Leskovec 2018].
- **Interpretabilidade**: O uso de mecanismos de atenção na GNN permite a extração de *scores* de importância, fornecendo insights sobre quais partes do subgrafo foram mais relevantes para a geração do *graph token*.
- **Eficiência**: O módulo é projetado para uma implementação otimizada, suportando processamento paralelo e em lote (*batching*) para garantir baixa latência.

Saídas Conceituais As principais saídas produzidas por este módulo são:

- **Graph Token**: A representação vetorial densa do subgrafo refinado, dimensionalmente compatível com o LLM.
- **Mapas de Atenção**: Distribuições de pesos que indicam a importância relativa de cada entidade e relação, úteis para a interpretabilidade da arquitetura.

O *graph token* produzido encapsula em uma forma compacta e processável toda a riqueza informacional do subgrafo recuperado e enriquecido. Esta representação serve como entrada de alta qualidade para o módulo de geração textual, onde o LLM utiliza tanto a pergunta original quanto o contexto estrutural enriquecido para gerar a resposta final ao usuário.

Módulo de Geração Textual

O **Módulo de Geração Textual** constitui o módulo conceitual responsável pela síntese final da resposta, integrando a pergunta original do usuário com o contexto estruturado e refinado produzido pelas fases anteriores. Sua arquitetura é fundamentada em princípios de modularidade, eficiência e adaptabilidade, permitindo a utilização de diferentes modelos de linguagem e estratégias de formatação de contexto.

Entradas Conceituais. As principais entradas deste módulo são:

- **Pergunta Original**: A consulta do usuário em linguagem natural.
- **Graph Token**: A representação vetorial densa do subgrafo refinado.
- **Representações Textuais das Evidências**: As triplas e descrições de entidades que compõem o contexto validado.
- **Parâmetros de Configuração**: Especificações sobre o modelo de linguagem, estratégia de *prompting* e limitações da resposta.

Processo Conceitual de Geração. O processo de geração textual é estruturado nas seguintes etapas:

1. **Construção do Prompt**: O contexto estruturado é formatado a partir das representações textuais das evidências. O *graph token* é incorporado ao *prompt* de acordo com a estratégia de integração adotada (e.g., concatenação, prefixo ou atenção cruzada), permitindo que a estrutura relacional e as inferências lógicas influenciem o processo de geração. Por fim, a pergunta original é adicionada.
2. **Inferência do LLM**: O *prompt* formatado é submetido a um modelo de linguagem, que realiza a inferência para gerar a resposta.
3. **Pós-processamento da Resposta**: A saída bruta do LLM pode ser processada para garantir consistência formal e aderência às especificações de saída.

Estratégias de Integração com LLMs. A arquitetura suporta múltiplas estratégias conceituais de integração, oferecendo flexibilidade para diferentes cenários:

- **Concatenação Contextual (*Prompt Tuning*)**: Abordagem padrão onde o *graph token* é tratado como parte do contexto textual do *prompt*.
- **Injeção de Prefixos (*Prefix Tuning*)**: Técnica avançada onde o *graph token* atua como um prefixo condicionante, influenciando o comportamento generativo do modelo.
- **Atenção Cruzada (*Cross-Attention*)**: Para modelos que suportam, a arquitetura prevê o uso de mecanismos que permitam ao LLM consultar dinamicamente as representações estruturais durante a geração.

Características Arquiteturais. O módulo apresenta as seguintes características distintivas:

- **Adaptabilidade de Modelos**: Suporte conceitual para diferentes arquiteturas de LLM, desde modelos compactos até versões de grande escala.
- **Otimização de Latência**: Previsão de mecanismos de *cache* para armazenar resultados de consultas recorrentes.
- **Escalabilidade**: Design preparado para processamento em lote e distribuição de carga, visando a implementação em ambientes de produção.

Saída Conceitual. A saída final do módulo é um texto em linguagem natural que constitui a resposta definitiva à pergunta do usuário, formatada para máxima clareza e utilidade.

4.3.4 Síntese da Arquitetura: Integração, Otimização e Explicabilidade

A eficácia da arquitetura proposta não reside apenas na qualidade individual de seus módulos, mas fundamentalmente na **sinergia entre as três fases operacionais**. A

integração otimizada do pipeline completo garante que a informação flua de maneira eficiente e coerente desde a recuperação inicial até a geração da resposta final, maximizando tanto a qualidade dos resultados quanto a eficiência computacional do sistema.

Esta integração é facilitada por um conjunto de **interfaces padronizadas** e **protocolos de comunicação** entre os módulos, que asseguram a interoperabilidade e permitem otimizações globais do pipeline. A arquitetura do *pipeline* completo também incorpora **estratégias de otimização transversais**, incluindo técnicas de paralelização seletiva, reutilização de computações intermediárias e balanceamento dinâmico de recursos computacionais. Estas otimizações são especialmente importantes para manter a competitividade do sistema em termos de latência e custo, objetivos centrais da proposta de uma solução "leve" e eficiente.

Finalmente, a integração completa do *pipeline* habilita funcionalidades avançadas como **explicabilidade end-to-end**, onde o sistema pode fornecer justificativas detalhadas sobre como uma resposta foi construída, rastreando as contribuições de cada uma das evidências extraídas e processadas. Esta capacidade é fundamental para aplicações que requerem transparência e auditabilidade, características cada vez mais valorizadas em sistemas de IA para uso crítico.

CAPÍTULO 5

O Framework ARANDU: Implementação e Detalhes Técnicos

Este capítulo dedica-se à materialização da arquitetura conceitual para Geração Aumentada por Recuperação (RAG) neuro-simbólica, apresentada no Capítulo 4. O objetivo é detalhar as decisões de engenharia, as ferramentas de software e os algoritmos específicos que, em conjunto, dão vida à proposta teórica, resultando em um sistema funcional, testável e reprodutível.

Como apresentado no Capítulo 1, esta implementação de referência recebe a denominação **ARANDU** (*Augmented Retrieval for Advanced Neuro-Symbolic Driven Understanding*). Ela é disponibilizada como uma contribuição de código aberto, através do repositório: http://github.com/otaviocx/arandu.

Para elucidar sua construção, este capítulo está organizado da seguinte forma: na Seção 5.1 são apresentados os módulos e componentes principais do framework ARANDU, com ênfase na arquitetura de software e na divisão entre preparação offline e execução online. A Seção 5.2 detalha o *stack* tecnológico e as principais bibliotecas utilizadas em cada camada do sistema. Em seguida, a Seção 5.3 descreve a implementação dos componentes de preparação offline, enquanto a Seção 5.4 aprofunda o pipeline de execução online, detalhando cada módulo da arquitetura. O capítulo se encerra com a especificação do componente de geração final (Seção 5.4.5).

5.1 Módulos e Componentes do Framework ARANDU

A implementação do framework ARANDU segue os princípios de design de software de alta modularidade e baixo acoplamento, refletindo diretamente a estrutura conceitual de fases e módulos descrita no Capítulo 4. A arquitetura de software foi dividida em dois conjuntos principais de componentes: aqueles responsáveis pela etapa de **preparação offline** e aqueles que constituem o **pipeline de execução online**.

A etapa de preparação, cujo diagrama de componentes é apresentado na Figura 5.1, é orquestrada por um módulo de **Ingestão** (`Ingestor`). Este componente utiliza

Carregadores (`DataLoaders`) para abstrair as fontes de dados e interage com componentes de **Armazenamento** (`Storage`) e um **Minerador de Regras** (`RuleMiner`) para gerar os artefatos necessários para a execução online. A utilização de uma interface de armazenamento abstrata garante a flexibilidade para a integração com diferentes tecnologias de bancos de dados vetoriais e de grafos.

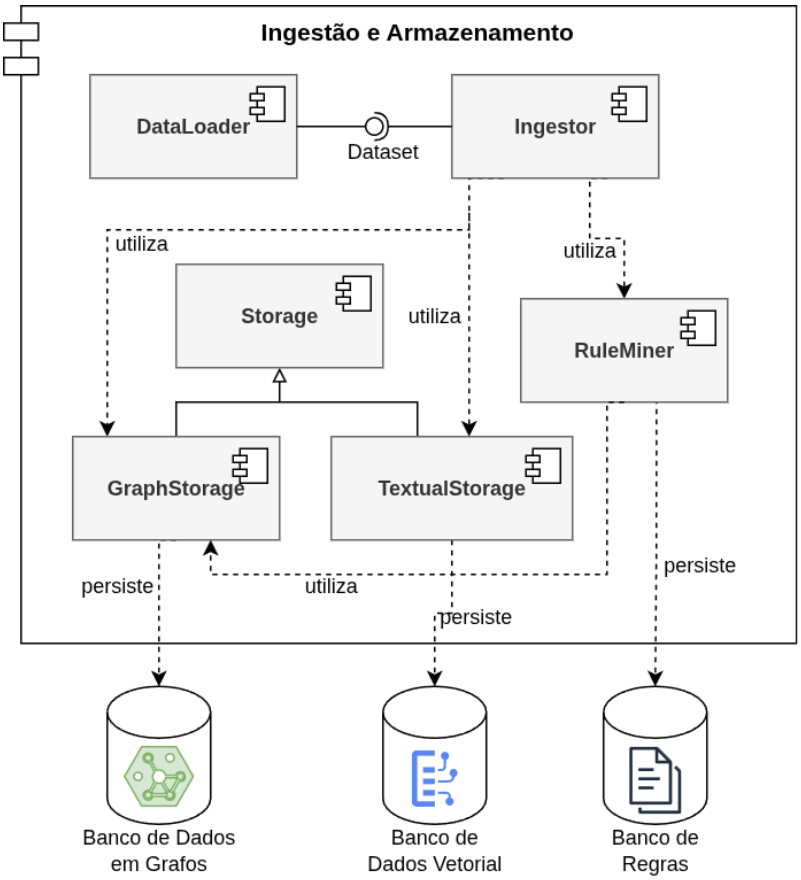

Figura 5.1: Diagrama de componentes da etapa de Ingestão e Armazenamento (Offline).

O pipeline de execução online, detalhado na Figura 5.2, é gerenciado por um **Orquestrador de Pipeline** (`PipelineOrchestrator`). Este componente central coordena uma sequência de módulos de serviço, cada um encapsulando a lógica de um módulo da arquitetura. O fluxo inicia com o **Recuperador Textual** (`TextRetriever`), passa pelo **Recuperador Estruturado** (`GraphRetriever`), pelo **Enriquecedor Lógico** (`LogicalEnricher`) e pelo **Ranqueador de Caminhos** (`PathRanker`), e culmina no **Modelo de Soft Prompt com Grafo** (`SoftPromptGraphModel`), que é responsável por interagir com o LLM e gerar a resposta final. Este design garante que cada etapa do processo seja um componente independente e substituível.

Tendo apresentado a estrutura modular do framework, a seção seguinte detalha as principais ferramentas e bibliotecas de software que foram escolhidas para implementar cada um desses componentes.

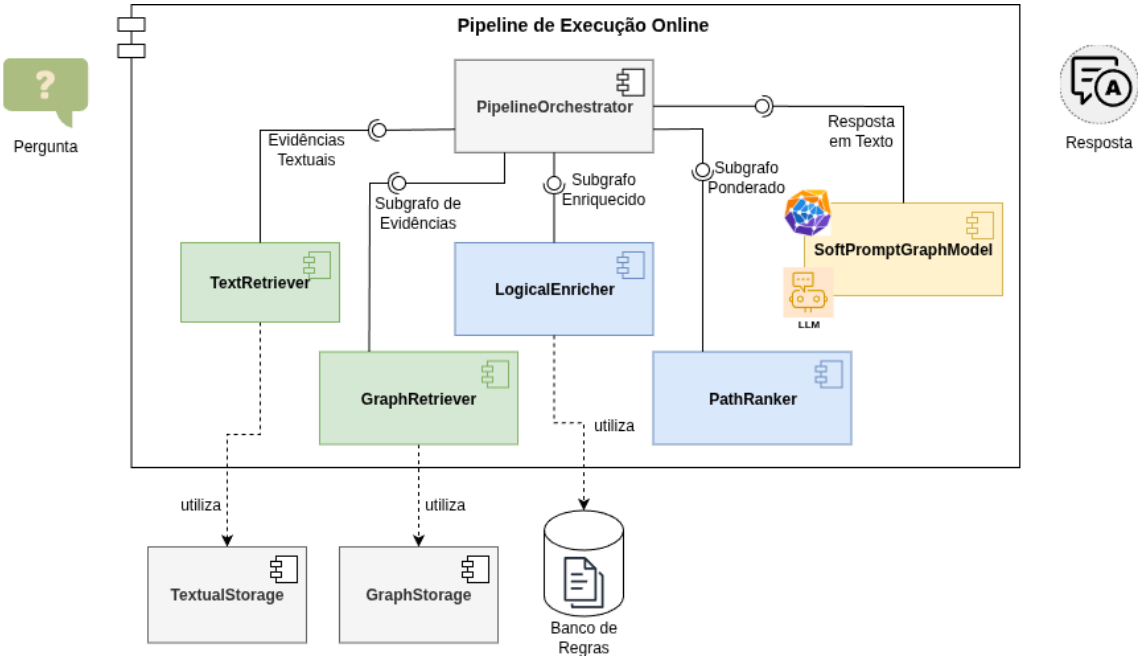

Figura 5.2: Diagrama de componentes do *Pipeline* de Execução Online.

5.2 Ferramentas e Tecnologias Adotadas

A materialização do framework ARANDU foi viabilizada pela seleção estratégica de um conjunto de bibliotecas e ferramentas de código aberto, alinhadas aos princípios de eficiência, modularidade e reprodutibilidade. Esta seção detalha as principais tecnologias empregadas em cada camada do framework, desde o processamento de linguagem natural até o armazenamento e o raciocínio em grafos.

Linguagem e Framework de Base A base do framework foi desenvolvida integralmente em **Python** (versão 3.9 ou superior), devido à sua maturidade e ao vasto ecossistema de bibliotecas para ciência de dados e inteligência artificial. Como framework de aprendizado profundo, foi adotado o **PyTorch** [Paszke et al. 2019], uma biblioteca de computação tensorial com forte aceleração GPU e suporte para diferenciação automática, escolhida por sua flexibilidade, API intuitiva e forte suporte da comunidade acadêmica para a implementação de arquiteturas neurais customizadas.

Modelos de Linguagem e Embeddings Para a interação com Grandes Modelos de Linguagem (LLMs), a biblioteca **Hugging Face Transformers** foi o pilar central, utilizada tanto para acessar modelos pré-treinados quanto para a geração final de respostas. Como LLMs de base para a etapa de geração (Fase III), foram selecionados os modelos **Llama 3.1 8B** e **Llama 3.2 3B**, que representam diferentes pontos no equilíbrio entre capacidade de geração e eficiência computacional. O **Llama 3.1 8B** proporciona capacidades

aprimoradas de raciocínio e compreensão textual com moderado aumento de custos computacionais, enquanto o **Llama 3.2 3B** destaca-se pela máxima eficiência, sendo ideal para cenários com restrições de recursos. Esses modelos foram escolhidos por estarem alinhados ao objetivo de um sistema "leve".

Para a codificação de textos em vetores (Módulo de Recuperação Textual), foram utilizados os modelos `all-MiniLM-L6-v2` [Wang et al. 2020] e `all-mpnet-base-v2` [Song et al. 2020] da biblioteca *sentence-transformers* [Reimers e Gurevych 2020]. O modelo `all-MiniLM-L6-v2` oferece um excelente equilíbrio entre desempenho e eficiência, gerando embeddings de 384 dimensões com processamento rápido, sendo ideal para cenários com restrições computacionais. Por sua vez, o `all-mpnet-base-v2` produz embeddings de 768 dimensões e apresenta desempenho superior em tarefas de recuperação semântica, embora demande maior capacidade computacional. Ambos os modelos têm se destacado pela sua ampla adoção na comunidade científica e desempenho consistente em *benchmarks* de recuperação semântica [1] , sendo considerados referências complementares para tarefas de busca e similaridade textual.

Redes Neurais para Grafos A implementação das arquiteturas GNN (Módulo de Representação Neural) e sua integração com as LLMs foi construída sobre a biblioteca **PyTorch Geometric (PyG)** [Fey e Lenssen 2019], uma extensão especializada do PyTorch que oferece implementações otimizadas de algoritmos de aprendizado em grafos e operações de processamento estrutural. A escolha da **Graph Attention Network v2 (GATv2)** [Brody, Alon e Yahav 2022] como arquitetura de GNN de referência foi motivada por seu mecanismo de atenção aprimorado, que corrige limitações da GAT original ao permitir que a função de atenção seja verdadeiramente dinâmica para cada aresta, possibilitando ponderar de forma mais eficaz a importância das evidências no subgrafo recuperado, uma característica importante para o raciocínio contextualizado.

Armazenamento e Recuperação Para a implementação da recuperação híbrida, duas tecnologias foram empregadas. O armazenamento e a busca vetorial (densa) foram gerenciados pelo **ChromaDB** [CHROMA 2025], um banco de dados vetorial de código aberto otimizado para buscas ANN de baixa latência.

A persistência e manipulação do grafo de conhecimento foram realizadas utilizando o **Neo4j** [Neo4j, Inc. 2025], um sistema de gerenciamento de banco de dados orientado a grafos que armazena dados em nós e relacionamentos, permitindo consul-

[1] No *benchmark* MTEB (*Massive Text Embedding Benchmark*), o modelo `all-mpnet-base-v2` está entre os modelos de pontuação mais alta para o seu tamanho (109M), bem como o `all-MiniLM-L6-v2` está entre os melhores no tamanho até 22M de parâmetros. Disponível em: https://huggingface.co/spaces/mteb/leaderboard

tas eficientes através da linguagem Cypher e oferecendo operações otimizadas para travessia e análise de estruturas de grafos. A linguagem Cypher é uma linguagem declarativa de consulta, caracterizada por uma sintaxe intuitiva que emprega padrões ASCII-art para representar grafos. A Cypher serviu como base para o desenvolvimento da **Graph Query Language (GQL)**[ISO/IEC 2024], um padrão ISO em desenvolvimento (ISO/IEC 39075:2024) que visa padronizar as consultas em bancos de dados de grafos, oferecendo uma sintaxe unificada para diferentes sistemas de gerenciamento de grafos.

A busca lexical (esparsa) foi implementada utilizando a biblioteca **bm25s** [Lù 2024], uma biblioteca que implementa o BM25[Robertson e Zaragoza 2009] em Python, permitindo classificar documentos com base em uma consulta. O BM25 é uma função de classificação amplamente utilizada para tarefas de recuperação de texto e é um componente central de serviços de busca como o Elasticsearch[2]. Índices de *full-text search* do Neo4j também foram utilizados, visando à comparação entre ambas as abordagens.

Raciocínio Simbólico e Orquestração Lógica A capacidade de raciocínio lógico da arquitetura, correspondente aos Módulos de Mineração de Regras Lógicas, Enriquecimento Lógico e Ponderação de Caminhos, é materializada em duas etapas distintas, cada uma empregando ferramentas especializadas.

Para a etapa de preparação offline (Módulo de Mineração de Regras Lógicas), foi utilizada a ferramenta de código aberto **AMIE3** [Lajus, Galárraga e Suchanek 2020]. Este sistema é especializado na descoberta de regras de Horn a partir de Grafos de Conhecimento, permitindo a mineração de um banco de regras de alta confiança que captura inferências lógicas latentes na estrutura dos dados.

Para a etapa de execução online (Módulos de Enriquecimento Lógico e Ponderação de Caminhos), o subgrafo de evidências recuperado é temporariamente transformado em um grafo RDF em memória. Isso permite a execução de consultas estruturadas e operações de inferência lógica.

O *Resource Description Framework* (**RDF**) [Consortium et al. 2014] é um modelo de dados padrão W3C[3] para representação de informações na Web, baseado em triplas com a forma `(sujeito, predicado, objeto)`. No contexto RDF, recursos são identificados por URIs (Uniform Resource Identifiers), predicados representam propriedades ou relacionamentos, e objetos podem ser recursos ou valores literais. Esta repre-

[2]Elasticsearch é uma plataforma de busca e análise distribuída baseada no Apache Lucene, amplamente utilizada para indexação e recuperação de texto em larga escala. Disponível em: https://www.elastic.co/elasticsearch/

[3]O World Wide Web Consortium (W3C) é um consórcio internacional que desenvolve padrões abertos para garantir o crescimento de longo prazo da Web. Disponível em: https://www.w3.org/

sentação é particularmente adequada para a materialização de Grafos de Conhecimento, pois oferece uma semântica formal bem estabelecida e suporte nativo para operações de inferência lógica.

Para consultar e manipular dados RDF, o padrão **SPARQL** [Harris e Seaborne 2013] (*SPARQL Protocol and RDF Query Language*) foi adotado. SPARQL é uma linguagem de consulta declarativa que permite extrair, filtrar e manipular dados armazenados em formato RDF. Sua sintaxe é baseada em padrões de triplas, onde variáveis (precedidas por `?`) podem ser utilizadas para capturar valores desconhecidos, permitindo consultas expressivas como inferência de novos relacionamentos e navegação em caminhos complexos no grafo.

A implementação desta funcionalidade foi realizada com a biblioteca **Oxigraph** [Tanon 2025]. Trata-se de uma ferramenta de código aberto, escrita em Rust com *bindings* para Python. Ela oferece um motor eficiente para armazenamento RDF em memória e execução de consultas SPARQL. A escolha foi motivada por sua performance superior e conformidade com os padrões W3C. Sua capacidade de processar consultas complexas com baixa latência é essencial para a etapa de expansão do subgrafo e ponderação dos caminhos.

5.3 Implementação da Preparação Offline

A eficácia do pipeline de execução online do framework ARANDU é intrinsecamente dependente da qualidade dos artefatos de conhecimento gerados na etapa de preparação offline. Este processo, executado uma única vez sobre o corpus de dados, é responsável por transformar o conhecimento bruto, seja ele textual ou estruturado, em representações otimizadas para recuperação e raciocínio em tempo real. A implementação desta fase foi dividida em um conjunto de componentes modulares, cada um responsável por uma etapa específica da preparação, desde o carregamento dos dados até a indexação e o enriquecimento lógico.

O fluxo de preparação é orquestrado por um componente de ingestão (`Ingestor`) que coordena a transformação dos dados brutos, abstraídos por um `DataLoader`, em três artefatos principais: um índice vetorial para busca semântica, um grafo de conhecimento persistido para navegação estrutural e um banco de regras para a orquestração lógica. Opcionalmente, um índice lexical para busca por palavras-chave pode ser gerado. As subseções seguintes detalham a implementação de cada um dos componentes responsáveis pela criação destes artefatos.

5.3.1 Carregamento e Ingestão de Dados

A primeira etapa na implementação da preparação offline consiste no carregamento e na ingestão dos dados brutos que constituem a base de conhecimento (Módulo de Abstração do Conjunto de Dados). Para garantir a modularidade e a adaptabilidade do framework a diferentes fontes e formatos de dados, este processo foi dividido em dois componentes de software distintos: o **Carregador** (`DataLoader`) e o **Ingestor** (`Ingestor`), conforme ilustrado no diagrama de componentes da Figura 5.1.

O componente **`DataLoader`** é projetado sob o princípio da separação de interesses, encapsulando toda a lógica necessária para ler e analisar um formato de dataset específico. Sua responsabilidade é traduzir a estrutura de dados original de uma fonte (seja ela um conjunto de arquivos JSON, CSV ou um dump de banco de dados) para um formato de representação interna e padronizado utilizado pelo ARANDU. Para cada novo dataset, uma implementação específica do `DataLoader` é necessária, permitindo que o restante do framework permaneça completamente agnóstico em relação à origem dos dados. A Figura 5.3 apresenta as implementações concretas para essa interface. Os componentes em azul são as implementações de referência, utilizadas durante os experimentos. Esta abordagem de projeto, análoga ao padrão de software *Adapter*, é fundamental para a extensibilidade do framework.

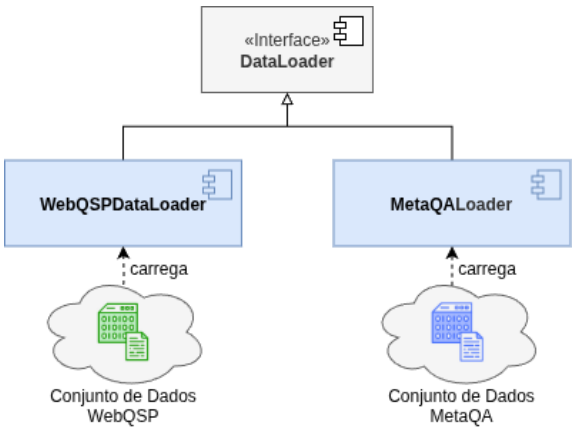

Figura 5.3: Diagrama de componentes com as implementações concretas da interface `DataLoader`. Uma para cada *dataset* utilizado.

O componente `DataLoader` implementa uma interface padronizada baseada em *generators* do Python, uma estrutura de dados que permite a iteração eficiente sobre grandes conjuntos de dados sem a necessidade de carregar todos os elementos em memória simultaneamente. Um *generator* é um tipo especial de iterador em Python que produz itens sob demanda (*lazy evaluation*), utilizando a palavra-chave `yield` para retornar valores sequencialmente. Esta abordagem é particularmente vantajosa quando

se trabalha com datasets extensos, pois minimiza o consumo de memória e permite o processamento incremental dos dados.

A interface do `DataLoader` é definida como um *generator* que produz itens do tipo `GraphQAItem`, uma estrutura de dados padronizada que encapsula todos os elementos necessários para uma instância de pergunta-resposta contextualizada por um grafo, conforme apresentado no Código 5.1:

Código 5.1 Estrutura de dados padronizada para itens de pergunta-resposta contextualizados por grafo

```
@dataclass
class GraphQAItem:
    id: int
    question: str
    str_answer: str
    answers: List[str]
    graph: torch_geometric.data.Data
    context: Optional[str]
```

Esta estrutura padronizada contém o identificador único da instância (`id`), a pergunta formulada em linguagem natural (`question`), a resposta em formato textual (`str_answer`), no caso de uma pergunta com múltiplas respostas, uma lista de possíveis respostas válidas (`answers`), o tipo da pergunta quando disponível (`type`) - por exemplo qualificação da pergunta em relação ao tipo de raciocínio que ela demanda (quantitativo, comparativo, etc.), e o subgrafo de conhecimento associado representado como uma lista de triplas no formato `(entidade sujeito, predicado/relação, entidade objeto)` (`graph`). A utilização de *generators* combinada com esta estrutura padronizada garante que o `DataLoader` possa processar datasets de qualquer tamanho de forma eficiente, mantendo a flexibilidade necessária para adaptar-se a diferentes formatos de origem sem impactar o desempenho ou a escalabilidade do sistema.

Uma vez que os dados são padronizados pelo `DataLoader`, o componente **`Ingestor`** assume o controle do fluxo de preparação. Atuando como um orquestrador, o `Ingestor` recebe os dados padronizados e coordena as chamadas para os diversos componentes de armazenamento e enriquecimento. Ele é responsável por iterar sobre as entidades e relações, invocando o `TextualStorage` para a criação dos índices de busca, e o `GraphStorage` para a persistência da estrutura do grafo. Adicionalmente, é o `Ingestor` que aciona o `RuleMiner` para a mineração de regras, garantindo que todos os artefatos de conhecimento sejam gerados de forma coesa e sequencial.

O componente `Ingestor` recebe como parâmetros as implementações específicas dos componentes de armazenamento (`TextualStorage` e `GraphStorage`) e mineração (`RuleMiner`), seguindo o princípio da injeção de dependência (*dependency injection*).

A injeção de dependência é um padrão de design que promove o desacoplamento entre componentes de software, onde as dependências de um objeto são fornecidas externamente em vez de serem criadas internamente pelo próprio objeto. Essa abordagem torna o sistema mais modular, testável e flexível, permitindo que diferentes implementações de cada componente sejam facilmente substituídas sem modificar o código do `Ingestor`. Por exemplo, diferentes implementações de `TextualStorage` (como Chroma, Pinecone ou FAISS) podem ser utilizadas, desde que implementem a interface padronizada esperada pelo orquestrador.

5.3.2 Implementação do Armazenamento Híbrido

A persistência e a organização eficiente da base de conhecimento constituem fundamentos essenciais para o desempenho do pipeline de recuperação. O framework ARANDU adota uma arquitetura de armazenamento híbrida, concebida para suportar múltiplas modalidades de acesso aos dados: tanto o acesso estruturado, baseado em grafos, quanto o acesso a conteúdo textual. Conforme apresentado na Figura 5.1, essa funcionalidade é encapsulada em camadas de abstração de armazenamento (`Storage`), para as quais são desenvolvidas implementações concretas, cada uma voltada a uma finalidade específica. A Figura 5.4 apresenta as implementações concretas para essa interface. Os componentes em azul são as implementações de referência, utilizadas durante os experimentos.

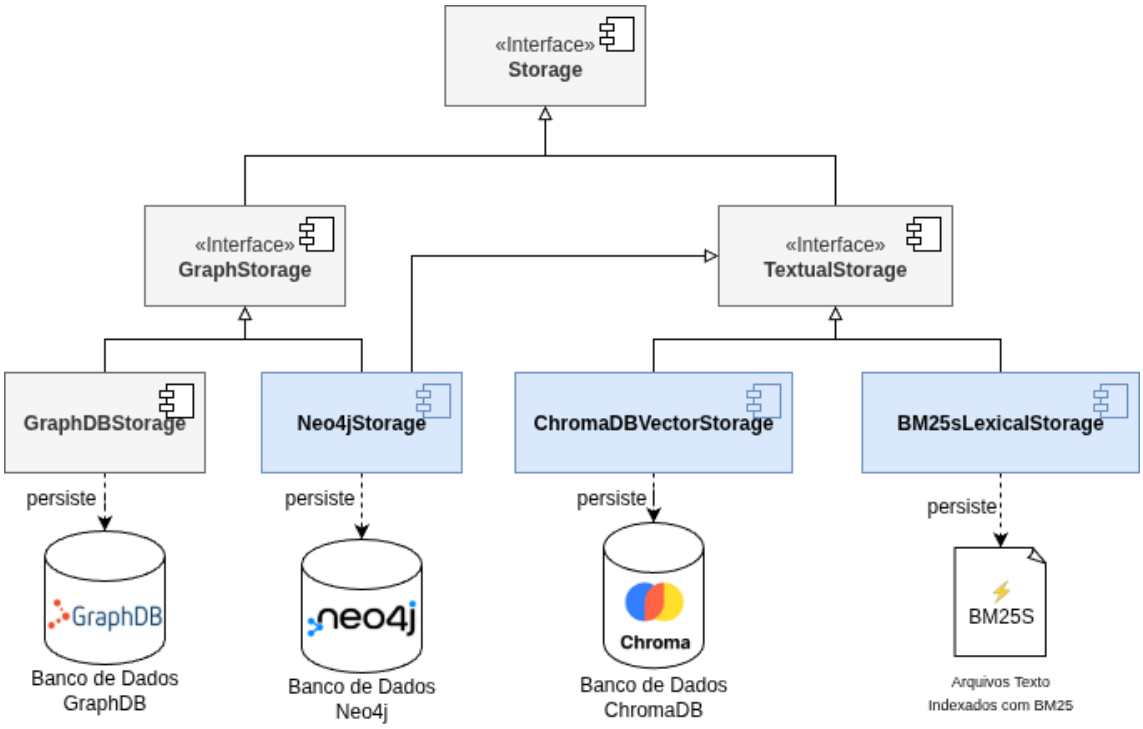

Figura 5.4: Implementações concretas para a interface `Storage`.

GraphStorage O componente **`GraphStorage`** é responsável por manter a representação estrutural do Grafo de Conhecimento. A tecnologia de referência selecionada é o **Neo4j** [Neo4j, Inc. 2025], cuja principal atribuição consiste em armazenar o conjunto de triplas `(sujeito, predicado, objeto)` e prover acesso eficiente à topologia do grafo por meio de consultas na linguagem **Cypher**. Para tal, foi desenvolvida a classe `Neo4jStorage`, que implementa a interface padrão definida por `GraphStorage` e utiliza o Neo4j como *backend*. Adicionalmente, foi desenvolvida uma implementação baseada no GraphDB[4], denominada `GraphDBGraphStorage`. Ao longo do desenvolvimento do framework e durante a realização dos experimentos, constatou-se que o Neo4j apresentou desempenho superior em termos de latência e escalabilidade, além de oferecer funcionalidades adicionais, como a indexação vetorial. Por esses motivos, o Neo4j foi adotado como implementação de referência para o `GraphStorage`.

TextualStorage: Uma Interface Unificada para Busca Textual Com o objetivo de suportar tanto a recuperação semântica (densa) quanto a lexical (esparsa), foi projetada uma interface unificada denominada **`TextualStorage`**. Essa interface, implementada como uma classe abstrata em Python, define um "contrato" comum para qualquer mecanismo de busca textual, assegurando que tanto o componente `Ingestor` quanto o componente `TextRetriever` (a ser detalhado na Seção 5.4.1) possam interagir com diferentes tecnologias de indexação e busca de maneira agnóstica. A interface padroniza métodos essenciais, tais como `add(items)`, `build_index()` e `query(text, topk)`, promovendo baixo acoplamento e modularidade no framework.

No contexto do ARANDU, foram desenvolvidas três implementações concretas para essa interface:

- **Implementação Vetorial (`ChromaDBVectorStorage`):** Esta classe implementa a interface `TextualStorage` para viabilizar a busca semântica. Internamente, utiliza o banco de dados vetorial **ChromaDB**. No método `add`, os itens textuais são primeiramente convertidos em *embeddings*, empregando modelos como o `all-MiniLM-L6-v2` [Wang et al. 2020], antes de serem persistidos no ChromaDB. O método `query` realiza uma busca aproximada por vizinhos mais próximos (ANN), identificando os vetores mais similares à consulta.
- **Implementação Lexical (`BM25sLexicalStorage`):** Esta classe implementa a mesma interface `TextualStorage` para busca por palavras-chave. Utiliza, internamente, a biblioteca **bm25s** para construir um índice invertido a partir dos itens textuais. O método `build_index` finaliza a construção desse índice, enquanto o

[4]GraphDB é um banco de dados de grafos fundamentado em RDF e SPARQL, desenvolvido pela Ontotext: https://graphdb.ontotext.com/

método `query` aplica o algoritmo BM25 [Robertson e Zaragoza 2009] para classificar e retornar os documentos mais relevantes, com base na correspondência de termos.

Cabe ressaltar que, em virtude da capacidade nativa do Neo4j de suportar indexação vetorial, a classe `Neo4jStorage`, aproveitando a flexibilidade da herança múltipla em Python, foi especializada para também herdar da interface `TextualStorage`. Dessa forma, ela pode ser utilizada tanto para o armazenamento estruturado do grafo quanto, opcionalmente, para a busca textual baseada em índice vetorial. Essa abordagem permite que um único componente ofereça ambas as funcionalidades, simplificando a arquitetura do framework em cenários onde a consolidação tecnológica é desejável, ao mesmo tempo em que explora as capacidades nativas do Neo4j para indexação de texto completo e busca por similaridade vetorial.

5.3.3 Implementação da Mineração de Regras

A capacidade do framework ARANDU de realizar um enriquecimento lógico sob demanda na etapa online depende de um artefato crucial gerado offline: um **Banco de Regras** de alta qualidade. A responsabilidade pela criação deste artefato é encapsulada no componente `RuleMiner` (Módulo de Mineração de Regras Lógicas).

Para a mineração das regras foi empregada a ferramenta de código aberto **AMIE3** [Lajus, Galárraga e Suchanek 2020], desenvolvida em Java. O AMIE3 (*Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases*) é um sistema avançado para mineração de regras de Horn em grandes Grafos de Conhecimento. Ele opera diretamente sobre arquivos de triplas RDF, identificando padrões recorrentes e extraindo regras lógicas de alta confiança que expressam inferências latentes na estrutura dos dados. O AMIE3 é capaz de lidar com grandes volumes de dados e incorpora métricas estatísticas como suporte, confiança padrão e confiança PCA (Partial Completeness Assumption) para filtrar regras relevantes.

Os principais padrões estruturais minerados pelo AMIE3 incluem:

Regras de Cadeia (Chain Rules) Este é o tipo mais comum de regra multi-salto, onde uma sequência de relações implica em uma nova relação de "atalho". Elas unem duas variáveis (e.g., `x` e `z`) através de uma ou mais variáveis intermediárias (e.g., `y`).

- **Estrutura:** $A(x, y) \wedge B(y, z) \Rightarrow C(x, z)$
- **Exemplo Prático:**

$$\text{nasceu\_em}(\text{Pessoa}, \text{Cidade}) \wedge \text{localizado\_em}(\text{Cidade}, \text{Pais}) \Rightarrow \text{tem\_nacionalidade}(\text{Pessoa}, \text{Pais})$$

Neste caso:

- A primeira parte indica que a pessoa nasceu em uma cidade.
- A segunda que essa cidade está em um país.
- A inferência é que a pessoa possui nacionalidade desse país.

Regras com Relação Faltante (*Triangle Rules*) Estas regras fecham um "triângulo" de relações. Elas sugerem que se uma entidade tem duas relações distintas, uma terceira relação pode ser inferida.

- **Estrutura:** $A(x,y) \wedge B(x,z) \Rightarrow C(y,z)$
- **Exemplo Prático:**

$$\text{casado\_com}(\mathsf{Pessoa1},\mathsf{Pessoa2}) \wedge \text{tem\_filho}(\mathsf{Pessoa1},\mathsf{Filho}) \Rightarrow \text{tem\_filho}(\mathsf{Pessoa2},\mathsf{Filho})$$

Neste caso:

- Se uma pessoa (Pessoa1) é casada com outra (Pessoa2)...
- ...e Pessoa1 tem um filho (Filho)...
- ...então Pessoa2 também tem esse mesmo filho.

Regras com Constantes São regras especializadas que se aplicam apenas a uma entidade específica (uma constante), em vez de a qualquer variável.

- **Estrutura:** $A(x,\text{Constante}) \Rightarrow B(x,\text{OutraConstante})$
- **Exemplo Prático:**

$$\text{localizado\_em}(\mathsf{Universidade},\mathsf{Goiania}) \Rightarrow \text{localizado\_em}(\mathsf{Universidade},\mathsf{Brasil})$$

Neste caso:

- Se a universidade está localizada em Goiânia...
- ...podemos inferir que ela está localizada no Brasil.

Muitas propriedades lógicas clássicas podem ser expressas como instâncias específicas desses padrões de regras de Horn, como por exemplo:

Regra de Simetria Afirma que se uma relação é verdadeira de x para y, ela também é verdadeira de y para x.

- **Estrutura:** $A(x,y) \Rightarrow A(y,x)$
- **Exemplo Prático:** cônjuge(Pessoa1, Pessoa2) $\Rightarrow$ cônjuge(Pessoa2, Pessoa1).

Regra de Inversão Similar à simetria, mas infere uma relação diferente na direção oposta.

- **Estrutura:** $A(x, y) \Rightarrow B(y, x)$
- **Exemplo Prático:** tem_filho(Pai, Filho) ⇒ tem_pai(Filho, Pai).

Regra de Transitividade Um caso especial de uma "Regra de Cadeia" onde todas as relações são iguais.

- **Estrutura:** $A(x, y) \wedge A(y, z) \Rightarrow A(x, z)$
- **Exemplo Prático:**

$$\text{parte\_de}(\text{Componente}, \text{Sistema}) \wedge \mathit{parte\_de}(\text{Sistema}, \text{Complexo}) \Rightarrow \mathit{parte\_de}(\text{Componente}, \text{Complexo})$$

É importante notar que o AMIE3 e mineradores semelhantes são otimizados para descobrir regras de implicação positiva. Propriedades como funcionalidade, injetividade ou regras de exclusão (com negação, ¬) são geralmente parte de uma ontologia formal (definida em linguagens como OWL) e são usadas para **verificação de consistência**, em vez de serem mineradas a partir de dados de instância.

A integração do AMIE3 ao framework ARANDU foi realizada por meio da classe `AMIE3RuleMiner`, que implementa a interface proposta pelo componente `RuleMiner`, como apresentado na Figura 5.5 . Essa classe é responsável por automatizar todo o processo de mineração. Ela executa o AMIE3 como um processo externo, fornecendo o grafo de conhecimento original em formato de triplas e capturando a saída gerada pela ferramenta. O AMIE3 produz como resultado um arquivo no formato `.tsv` (tab-separated values), um tipo de arquivo texto em que os campos de cada linha são separados por tabulações, facilitando a leitura e o processamento automatizado dos dados extraídos.

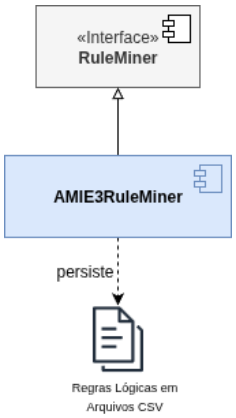

Figura 5.5: Implementação concreta para a interface `RuleMiner`.

O `AMIE3RuleMiner` é responsável por interpretar esse arquivo `.tsv`, realizando o parsing de cada linha para extrair as regras de Horn descobertas, juntamente com seus

respectivos scores estatísticos (suporte, confiança, confiança PCA, etc.). Cada regra é então convertida para uma estrutura padronizada denominada `GraphQARule`, que encapsula os seguintes campos principais: o corpo (`body`) e a cabeça `head` da regra (representados como listas de átomos ou triplas), e um campo de metadados (`metadata`), contendo métricas e detalhes adicionais da regra (suporte, confiança, qtd. de exemplos positivos, etc). Essa padronização permite que as regras mineradas sejam facilmente integradas ao pipeline do framework, servindo tanto para a materialização de novas inferências quanto para a orquestração lógica durante a execução online.

Para garantir a qualidade e a relevância das regras inferidas, uma etapa de filtragem é aplicada. Apenas regras que atendem a critérios pré-definidos de significância estatística são mantidas. A métrica de referência utilizada para esta filtragem foi a **Confiança PCA** (*Partial Completeness Assumption*), uma medida robusta que avalia a plausibilidade de uma regra. Outro hiperparâmetro ajustável nesse componente é a quantidade mínima de exemplos positivos (ou seja, o número de instâncias no grafo em que a regra pode ser aplicada com sucesso, indicando quantas vezes a regra foi observada como verdadeira nos dados).

5.4 Implementação do Pipeline de Execução Online

Com os artefatos de conhecimento devidamente preparados pela etapa offline, o framework ARANDU está apto a processar consultas em tempo real. Esta seção detalha a implementação do pipeline de execução online, um fluxo de componentes orquestrado que materializa as três fases da arquitetura (Recuperação, Refinamento e Geração) para transformar uma pergunta em linguagem natural em uma resposta fundamentada.

A implementação do pipeline é centrada no padrão de projeto *Orchestrator*[5]. Um componente central, o **PipelineOrchestrator**, atua como o "maestro" do fluxo, sendo responsável por gerenciar o estado da consulta e invocar cada um dos componentes de serviço na sequência correta. Conforme ilustrado na Figura 5.2, o orquestrador chama sequencialmente o `TextRetriever`, o `GraphRetriever`, o `LogicalEnricher`, o `PathRanker` e, finalmente, o `SoftPromptGraphModel`, passando a saída de um componente como entrada para o seguinte. Este design garante um baixo acoplamento entre os módulos, permitindo que cada um seja testado, mantido e substituído de forma independente. As subseções a seguir descrevem a implementação detalhada de cada um desses componentes de serviço.

[5] O padrão Orquestrador (*Orchestrator*), é um padrão de arquitetura de software que centraliza o controle do fluxo de execução entre múltiplos componentes ou serviços, promovendo baixo acoplamento e modularidade [Richards 2022].

Um aspecto fundamental do `PipelineOrchestrator` é sua **configurabilidade**, que permite ativar ou desativar dinamicamente qualquer um dos módulos do pipeline (Recuperação Textual, Recuperação Estruturada, Enriquecimento Lógica, Ponderação de Caminhos e Representação Neural via GNN) de acordo com as necessidades do caso de uso ou experimentação. Essa flexibilidade é viabilizada por meio de parâmetros de configuração centralizados, que controlam a execução condicional de cada componente do pipeline.

Por exemplo, é possível executar o pipeline com apenas um subconjunto das fases e módulos, como apenas a recuperação textual, ou apenas a recuperação estruturada, ou ainda pular a orquestração lógica. Todos os módulos, exceto o de geração, podem ser desativados simultaneamente: nesse cenário, a pergunta em linguagem natural é encaminhada diretamente ao modelo de linguagem (LLM), sem qualquer etapa prévia de recuperação ou aumento de contexto. Essa configuração é especialmente útil para fins de ablação, benchmarking e comparação direta entre diferentes estratégias de aumento de contexto, permitindo avaliar o impacto isolado de cada módulo na qualidade das respostas geradas.

Essa arquitetura modular e configurável reforça o baixo acoplamento entre os componentes do ARANDU, facilitando tanto a experimentação científica quanto a adaptação do framework a diferentes domínios e requisitos operacionais.

5.4.1 Implementação da Recuperação Textual (TextRetriever)

O componente **TextRetriever**, que materializa o Módulo de Recuperação Textual, é a porta de entrada do pipeline de execução e atua como um filtro de alta cobertura (*high-recall*). Como visto na Seção 4.3.1, sua principal responsabilidade é identificar um conjunto inicial de entidades e relações candidatas a partir da base de conhecimento. Para maximizar as chances de encontrar evidências relevantes, ele pode ativar uma estratégia de busca híbrida, combinando os pontos fortes da recuperação densa (semântica) e da recuperação lexical (esparsa).

A classe `ChromaDBBM25RRFTextRetriever` é a implementação padrão desse componente. O diagrama de componentes da Figura 5.6 apresenta as implementações concretas dos componentes de recuperação. Como é possível observar, esta implementação representa uma solução unificada, combinando as vantagens da busca semântica e lexical por meio da técnica de **Reciprocal Rank Fusion (RRF)** [Cormack, Clarke e Buettcher 2009]. O RRF é um método de fusão de *rankings* que integra múltiplas listas de resultados ordenadas, utilizando uma função de pontuação harmônica, definida como $\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + r(d)}$, em que $r(d)$ representa a posição do documento d no *ranking* r, e k é uma constante de suavização (tipica-

mente 60). Internamente, esta classe utiliza ambas as implementações de TextualStorage (`ChromaDBVectorStorage` e `BM25sLexicalStorage`), executando as consultas em paralelo e fundindo os resultados por meio do algoritmo RRF. Tal abordagem permite capturar tanto correspondências semânticas quanto lexicais, proporcionando uma recuperação mais robusta e abrangente.

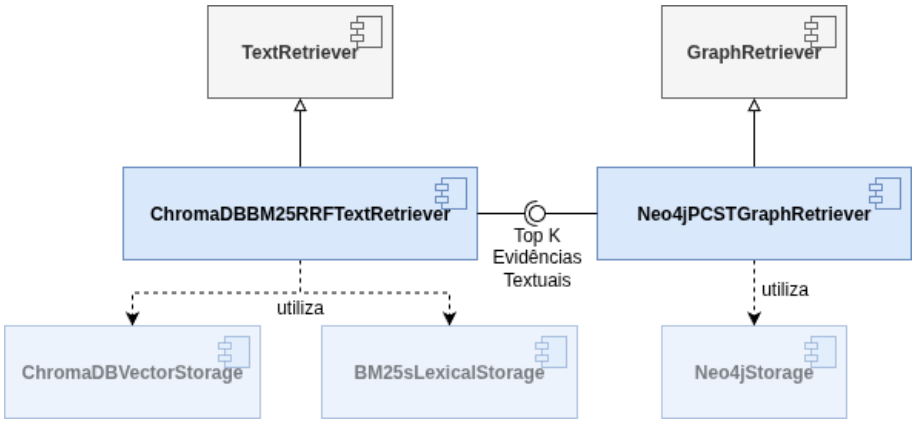

Figura 5.6: Implementações concretas para a fase de recuperação (interfaces `TextRetriever` e `GraphRetriever`).

Saída do Componente A saída final do `TextRetriever` é uma única lista unificada e reclassificada de evidências candidatas (entidades e relações), cada uma com seu *score* RRF. Esta lista, que representa o conjunto mais promissor de pontos de partida no grafo de conhecimento, é então retornada ao `PipelineOrchestrator`, que a encaminhará para o componente `GraphRetriever` (Módulo de Recuperação Estruturada) para a etapa de otimização de subgrafo.

5.4.2 Implementação da Recuperação de Subgrafo (GraphRetriever)

O componente **GraphRetriever**, que materializa o Módulo de Recuperação Estruturada, é responsável por transformar o conjunto de evidências textuais, fornecido pelo `TextRetriever`, em um subgrafo coeso e relevante para a pergunta. Em vez de simplesmente expandir a vizinhança dos nós candidatos, uma abordagem que poderia introduzir ruído significativo, o framework ARANDU adota uma abordagem de otimização formal. A implementação padrão (`Neo4jPCSTGraphRetriever`) modela esta tarefa como uma instância do problema **Prize-Collecting Steiner Tree (PCST)**, uma metodologia robusta para extração de subgrafos, inspirada diretamente pela abordagem proposta pelo G-Retriever [He et al. 2024]. Tal implementação também pode ser vista na Figura 5.6. Como é possível observar, essa implementação utiliza o componente `Neo4jStorage` para consultar no Grafo de Conhecimento. Adcionalmente, a implementação requer as Top-

K evidencias recuperadas pelo módulo `ChromaDBBM25RRFTextRetriever` (ou qualquer outra implementação de `TextRetriever`).

Para cada pergunta, uma nova instância do problema PCST é construída dinamicamente. O processo envolve a definição de prêmios, custos e terminais sobre uma porção do Grafo de Conhecimento principal, extraída do `GraphStorage`.

- **Prêmios (*Prizes*):** Os *scores* de relevância, calculados pelo `TextRetriever`, são diretamente mapeados como "prêmios" para cada nó e relação candidata. Os nós e relações retornados pelo `TextRetriever` recebem um prêmio correspondente à posição dele no *ranking*. Ou seja, se forem retornados os top 3 nós e as top 5 relações, o primeiro nó recebe o prêmio 3, o segundo, 2, e o terceiro, 1. Da mesma forma, a primeira relação recebe o prêmio 5 e assim por diante. Os nós e relações que não foram retornados recebem prêmio zero.
- **Custos (*Costs*):** O custo de inclusão de cada aresta (relação) no subgrafo é um hiperparâmetro fundamental, `edge_cost`. Um valor de custo elevado resulta em subgrafos menores e mais esparsos, focados nos candidatos de maior prêmio, favorecendo a precisão. Um custo baixo permite a inclusão de mais arestas e nós intermediários, criando subgrafos maiores e mais conectados, o que pode favorecer o *recall*. Este parâmetro permite ajustar o equilíbrio entre precisão e cobertura do subgrafo. Além disso, quando uma aresta está associada a um prêmio, seu custo é reduzido proporcionalmente ao valor do prêmio recebido, de modo que a aresta de maior prêmio tem custo zero. Isso incentiva a inclusão de conexões altamente relevantes no subgrafo extraído.

A implementação do PCST no ARANDU utiliza a biblioteca `fast_pcst`, disponível para Python. Essa biblioteca oferece uma implementação eficiente e otimizada do algoritmo, baseada em métodos aproximados de tempo quase-linear, sendo capaz de lidar com grafos de médio a grande porte de forma rápida. O `fast_pcst` recebe como entrada os nós, arestas, prêmios e custos, e retorna a árvore de Steiner ótima aproximada[6], tornando-se especialmente adequada para aplicações em pipelines de recuperação estruturada em tempo real.

Apesar da abordagem baseada em PCST ser a implementação padrão do componente `GraphRetriever`, o framework ARANDU foi projetado para ser flexível e modular, permitindo a adoção de diferentes estratégias de extração de subgrafos conforme as necessidades do domínio ou requisitos de desempenho. Duas outras implementações alternativas foram desenvolvidas e avaliadas:

[6] A **árvore de Steiner** é um subgrafo conexo que conecta um subconjunto de nós terminais de um grafo, minimizando o custo total das arestas utilizadas. No contexto do PCST, o objetivo é maximizar o valor total dos prêmios coletados menos o custo das arestas incluídas, permitindo a inclusão de nós intermediários não terminais para reduzir o custo total da conexão.

- ***Path Finding* (Busca de Caminhos Mais Relevantes):** Esta abordagem foi implementada no componente `Neo4jPathFindingGraphRetriever`, cuja principal função é construir o subgrafo de evidências a partir da identificação dos caminhos mais relevantes que conectam os nós e relações candidatos fornecidos pelo `TextRetriever`. O algoritmo realiza, para cada par de entidades candidatas, a busca dos caminhos mais curtos (*shortest paths*) no grafo de conhecimento, utilizando algoritmos clássicos como Dijkstra ou BFS, dependendo da ponderação das arestas.

  Para priorizar a relevância das conexões, os pesos das arestas são ajustados de acordo com os scores de relevância atribuídos pelo `TextRetriever`, de modo que caminhos que atravessam relações e entidades mais relevantes sejam favorecidos. Além disso, é imposto um limite superior ao comprimento dos caminhos (parâmetro configurável), prevenindo a explosão combinatória e garantindo a interpretabilidade do subgrafo resultante. O subgrafo final é composto pela união dos nós e arestas presentes nos caminhos selecionados, assegurando que as evidências estejam conectadas de forma direta e explicável. Esta implementação é especialmente adequada para cenários em que a transparência e a rastreabilidade das conexões entre as evidências são requisitos fundamentais.
- ***Beam Search* (Busca Heurística com Feixe):** Outra abordagem avaliada é a estratégia de extração de subgrafos baseada em *beam search*, formalizada no componente `Neo4jBeamSearchGraphRetriever`. Nesta implementação, a busca parte dos nós candidatos identificados pelo `TextRetriever` e expande iterativamente os caminhos mais promissores no grafo de conhecimento. Em cada iteração, apenas um número limitado de caminhos, determinado pelo hiperparâmetro *beam width*, é mantido. Assim, priorizando aqueles com maior score acumulado.

  O algoritmo opera da seguinte forma: para cada nó candidato inicial, são gerados todos os caminhos possíveis de comprimento unitário (i.e., uma aresta). Cada caminho recebe um score calculado como a soma dos scores dos nós e relações percorridos, conforme atribuído na etapa de recuperação textual. Em cada etapa subsequente, os caminhos existentes são expandidos por uma aresta adicional, e o score acumulado é atualizado. Após cada expansão, apenas os k caminhos de maior score (onde k é o *beam width*) são mantidos para a próxima iteração, descartando os demais. O processo é repetido até que um critério de parada seja atingido, como o comprimento máximo do caminho ou a convergência dos scores.

  Ao final da busca, o subgrafo de evidências é composto pela união dos nós e arestas presentes nos caminhos selecionados pelo feixe (*beam*). Esta abordagem permite explorar múltiplas rotas relevantes no grafo sem a necessidade de examinar exaustivamente todo o espaço de busca, sendo especialmente eficiente em grafos

de grande porte. O componente `BeamSearchGraphRetriever` foi projetado para ser parametrizável, permitindo o ajuste do *beam width*, do comprimento máximo dos caminhos e da função de score utilizada, de modo a equilibrar eficiência computacional e qualidade das evidências extraídas.

Essas alternativas ampliam a flexibilidade do ARANDU, permitindo a seleção da estratégia de recuperação estruturada mais adequada para cada cenário, seja priorizando precisão, cobertura, interpretabilidade ou eficiência computacional.

Saída do Componente Ao final da execução, o `GraphRetriever` retorna um conjunto de triplas, em que os nós e arestas são representados por tipos de dados próprios (`GraphNode` e `GraphEdge`), contendo atributos como ID, rótulo, descrição e score de relevância. Para representar o subgrafo completo, utiliza-se a classe `Data` do framework PyG (*PyTorch Geometric*), que serve como estrutura padrão para armazenar grafos, encapsulando as informações dos nós, arestas e seus atributos de forma eficiente e compatível com operações de aprendizado de máquina em grafos. O subgrafo resultante é então encaminhado pelo `PipelineOrchestrator` para a fase de orquestração lógica.

5.4.3 Implementação do Enriquecimento Lógico (LogicalEnricher)

A fase de Orquestração Lógica do Contexto é o núcleo do motor de raciocínio do ARANDU. Sua função é evidenciar, no subgrafo coeso vindo do Módulo de Recuperação Estruturada, caminhos de raciocínio enxutos, completos e precisos.

O processo é implementado em duas partes. Primeiro, é realizada uma **expansão lógica**, através do módulo de Enriquecimento Lógico (LogicalEnricher). As regras pré-mineradas (offline) são aplicadas para inferir novas triplas e completar caminhos de raciocínio no subgrafo. Em seguida, a **ponderação de relevância** é executada, através do módulo de Ponderação de Caminhos (PathRanker). Um algoritmo customizado ranqueia os caminhos e atribui pesos às arestas, refletindo sua importância contextual para a pergunta.

Para realizar a expansão lógica são utilizadas consultas SPARQL. Logo, a implementação padrão é denominada `SPARQLLogicalEnricher`. O diagrama de componentes da Figura 5.7 apresenta esse componente e suas interações. Este componente utiliza o **Banco de Regras** (representado por arquivos CSV), gerado pelo RuleMiner (Módulo de Mineração de Regras Lógicas), e aplica essas regras dinamicamente apenas sobre o pequeno subgrafo recuperado. A implementação utiliza um algoritmo que aplica iterativamente as regras ao subgrafo, carregado como um objeto `pyoxigraph.Store`, até que nenhuma nova tripla possa ser inferida.

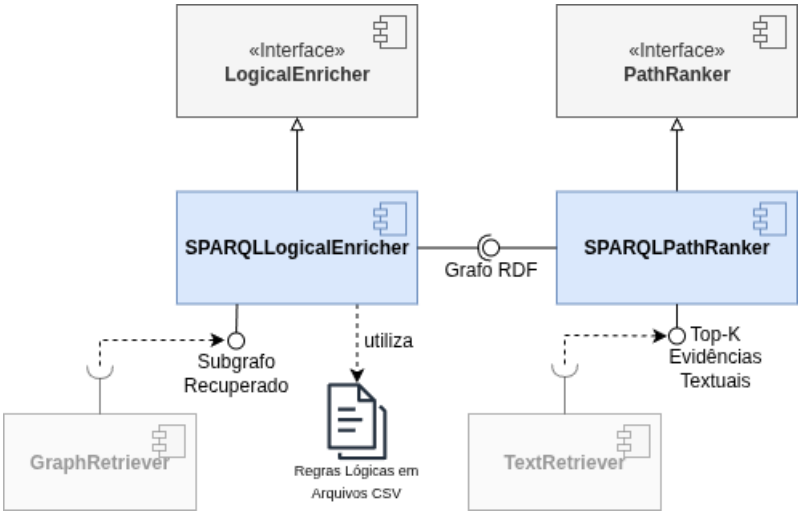

Figura 5.7: Diagrama de Componentes com implementações concretas para a fase de orquestração lógica (interfaces `LogicalEnricher` e `PathRanker`).

O Código ?? apresenta um template de consulta SPARQL utilizado para a expansão lógica do subgrafo. Este template é construído dinamicamente a partir das regras mineradas e aplicadas ao subgrafo de evidências. A consulta é composta por três partes principais:

1. **Declaração de PREFIX**: Define abreviações de *namespaces* para URIs. No template, a diretiva `PREFIX ex: <{EX}>` estabelece o *namespace* base para os recursos gerados, permitindo URIs compactas e consistentes.
2. **Cabeçalho CONSTRUCT**: Define o padrão de triplas que serão geradas como resultado da consulta. No template, a construção é feita a partir da variável `row['Head']`, que representa a cabeça da regra lógica aplicada. A função `parse_triple` é utilizada para transformar a representação da tripla em um formato adequado para a consulta SPARQL.
3. **Cláusula WHERE**: Especifica as condições que devem ser satisfeitas para que as triplas sejam geradas. Neste caso, a cláusula WHERE é composta por uma junção de triplas representadas por `row['Body']`, que corresponde ao corpo da regra lógica. Cada tripla no corpo é processada pela função `parse_triple` para garantir que esteja no formato correto para a execução da consulta.

O uso de placeholders `{EX}` e a função `parse_triple` tornam o template altamente parametrizável, permitindo a substituição dinâmica de partes da consulta com base nas regras específicas e no contexto do subgrafo. Isso possibilita a aplicação eficiente de inferências lógicas sobre o subgrafo em memória, enriquecendo-o com novas triplas derivadas das regras mineradas.

Código 5.2 Template SPARQL para expansão lógica do subgrafo

```
PREFIX ex: <{EX}>
CONSTRUCT {
    {parse_triple(row['Head'])} .
}
WHERE {
    {" . ".join([parse_triple(tri) for tri in row['Body']])}
}
```

Saída do Componente Ao final deste processo, o componente retorna o subgrafo de evidências enriquecido com novas triplas inferidas pelas regras lógicas. O grafo expandido mantém a estrutura original do subgrafo recuperado, acrescido das triplas derivadas logicamente. Este grafo logicamente enriquecido é representado como um grafo de triplas RDF, implementado através de um objeto `pyoxigraph.Store`. O subgrafo expandido é então encaminhado para o Módulo de Ponderação de Caminhos.

5.4.4 Implementação da Ponderação de Caminhos (PathRanker)

Com o subgrafo agora logicamente expandido, a segunda parte da orquestração visa ponderar a relevância de suas entidades e relações. O processo implementa um algoritmo determinístico de ranqueamento, dividido em três sub-etapas:

1. **Recuperação de Sementes de Relevância:** Primeiramente, o sistema reutiliza as evidências (nós e arestas) recuperadas pelo `TextRetriever`, na etapa de recuperação textual, e identifica no subgrafo as entidades e relações que possuem a maior similaridade semântica com a pergunta do usuário. Estes elementos de alta relevância servem como "sementes" ou "âncoras" para a descoberta de caminhos.
2. **Descoberta de Caminhos de Conexão:** Em seguida, é gerada uma consulta SPARQL programaticamente. O objetivo desta consulta é atuar como um motor de busca de caminhos sobre o grafo expandido em memória. A consulta busca por todos os caminhos que conectam as sementes de relevância identificadas na etapa anterior.
3. **Ranqueamento e Ponderação:** Por fim, os caminhos retornados pela consulta são avaliados por um algoritmo de pontuação. Este algoritmo atribui pesos aos caminhos com base em critérios como comprimento e relevância das sementes que conectam. Os pesos dos caminhos são então agregados para calcular um peso final para cada entidade e relação individual no subgrafo. Elementos que participam de múltiplos caminhos de alta pontuação recebem um peso maior.

Assim como o enriquecimento lógico, este módulo também utiliza a linguagem SPARQL. Logo, a implementação padrão é denominada `SPARQLPathRanker`. Esta im-

plementação e suas interações com outros componentes são apresentadas na Figura 5.7. O Código **??** apresenta um template de consulta SPARQL utilizado para a descoberta de caminhos de conexão. Este template é construído dinamicamente a partir das sementes de relevância identificadas na etapa anterior. A consulta é composta por três partes principais:

1. **Declaração de PREFIX**: Define o namespace base para os recursos do grafo, onde `{EX}` é substituído dinamicamente pelo namespace específico do subgrafo sendo analisado.
2. **Cláusula SELECT**: Especifica as variáveis a serem retornadas, representando os elementos de caminhos de até 2 arestas: nó inicial (`?start`), predicados (`?p1`, `?p2`), nó intermediário (`?n1`), e nó final (`?end`).
3. **Primeira subquery (UNION)**: Busca caminhos diretos de 1 aresta entre entidades, aplicando um filtro para garantir que a propriedade conectora (`?p1`) esteja entre as arestas mais relevantes identificadas na etapa anterior (`topk_edges_str`).
4. **Segunda subquery (UNION)**: Busca caminhos indiretos de 2 arestas, conectando entidades através de um nó intermediário (`?n1`). O filtro `OR` garante que pelo menos uma das propriedades do caminho (`?p1` ou `?p2`) esteja entre as mais relevantes.
5. **Filtros de integridade**: Aplicados à segunda subquery para evitar caminhos degenerados, garantindo que o nó inicial, intermediário e final sejam distintos entre si.
6. **Filtro global de relevância**: Aplicado a todos os caminhos gerados, garantindo que pelo menos um extremo do caminho (nó inicial ou final) esteja entre as entidades sementes de alta relevância (`topk_nodes_idx_str`) identificadas na etapa de recuperação textual.

É importante observar que o template apresentado no Código **??** demonstra a busca por caminhos de até 2 arestas (ou 2 saltos). Esta configuração se mostrou mais eficaz nos experimentos realizados. No entanto, o design modular do componente permite facilmente estender a consulta para contemplar caminhos de maior comprimento, simplesmente adicionando blocos UNION adicionais para 3, 4 ou mais saltos. Empiricamente, observamos que caminhos de 2 saltos oferecem o melhor equilíbrio entre cobertura de informações relevantes e controle de ruído contextual, evitando a inclusão de entidades demasiadamente distantes da questão original.

Após a descoberta de caminhos através da consulta SPARQL, é necessário quantificar a relevância de cada caminho identificado para a pergunta original. O sistema implementa um mecanismo de pontuação que combina as relevâncias individuais dos componentes do caminho (nós e arestas) para produzir uma pontuação agregada que reflete a importância contextual de cada caminho.

O processo de ponderação opera em três etapas distintas:

Código 5.3 Template SPARQL para descoberta de caminhos de conexão com até 2 arestas

```
PREFIX : <{EX}>

SELECT DISTINCT ?start ?p1 ?n1 ?p2 ?n2 ?p3 ?end
WHERE {{
{{
    # Caminho de 1 aresta
    ?start ?p1 ?end .
    FILTER (?p1 IN (
    :{topk_edges_str}
    ))
}}
UNION
{{
    # Caminho de 2 arestas
    ?start ?p1 ?n1 .
    ?n1 ?p2 ?end .
    FILTER (?p1 IN (
    :{topk_edges_str}
    ) || ?p2 IN (
    :{topk_edges_str}
    ))
    FILTER (?start != ?n1)
    FILTER (?start != ?end)
    FILTER (?n1 != ?end)
}}
# Este filtro  aplicado a todos os caminhos gerados pelos UNIONs
FILTER (?start IN (:{topk_nodes_idx_str}) || ?end IN
    ↪(:{topk_nodes_idx_str}))
```

1. **Normalização de pontuações base**: As entidades e relações previamente ranqueadas na etapa de recuperação textual recebem pontuações normalizadas inversamente proporcionais à sua posição no ranking. Especificamente, uma entidade ou relação na posição i de um ranking de tamanho n recebe uma pontuação $\text{score}(i) = \frac{n-i}{n}$, garantindo que elementos mais relevantes (posições menores) obtenham pontuações maiores.
2. **Cálculo de pontuação por caminho**: Para cada caminho identificado, a pontuação agregada é calculada como a soma das pontuações normalizadas de seus componentes. Para um caminho direto de 1 aresta entre nós s e e através da propriedade p_1, a pontuação é:

$$\text{score(caminho)} = \text{score}(s) + \text{score}(e) + \text{score}(p_1) \tag{5-1}$$

Para caminhos de 2 arestas que conectam s a e através de um nó intermediário n_1

via propriedades p_1 e p_2, a pontuação é:

$$\text{score(caminho)} = \text{score}(s) + \text{score}(e) + \text{score}(p_1) + \text{score}(p_2) \tag{5-2}$$

3. **Propagação de pontuações para componentes**: A pontuação calculada para cada caminho é propagada de volta para suas triplas constituintes, nós e arestas. Esta propagação permite que elementos que participem de múltiplos caminhos relevantes acumulem pontuações maiores, refletindo sua importância central na rede de evidências. Após a propagação, tanto as estruturas de nós quanto de triplas são reordenadas em ordem decrescente de pontuação. Esta reordenação é fundamental para a representação textual do grafo, pois garante que as informações mais relevantes sejam apresentadas prioritariamente no contexto fornecido ao LLM, otimizando assim a qualidade da geração subsequente.

Este mecanismo de ponderação garante que o subgrafo final preserve não apenas as conexões mais semanticamente relevantes, mas também quantifique explicitamente a importância de cada elemento para a pergunta original. As pontuações resultantes são posteriormente utilizadas pelo componente GNN para ponderar adequadamente a atenção durante a geração do *graph token*, criando uma ponte eficaz entre a relevância simbólica identificada pelo componente de orquestração lógica e a representação neural.

Saída do Componente Ao final deste processo, o componente retorna o subgrafo de evidências com dois enriquecimentos principais: (1) novas triplas inferidas pelas regras lógicas e (2) um atributo de peso (`score`) para cada aresta, refletindo sua importância contextual para a pergunta. Este grafo, logicamente enriquecido e ponderado, é representado como um objeto `Data` do framework PyG [Fey e Lenssen 2019] e é então encaminhado para o Módulo de Representação Neural via GNN.

5.4.5 Representação Neural e Geração Final (SoftPromptGraphModel)

A fase final do pipeline online, que abrange tanto a representação neural (Módulo de Representação Neural via GNN) quanto a geração da resposta (Módulo de Geração Textual via LLM), é encapsulada em um único componente de software: o **SoftPromptGraphModel**. Esta implementação, concebida como uma implementação padrão de uma classe `nn.Module` do **PyTorch** [Paszke et al. 2019], isola a única parte treinável do framework e materializa a ponte entre a evidência estruturada e a geração textual. Este design unificado simplifica a integração e permite que o componente se comporte de maneira padronizada nos ciclos de treinamento e inferência.

Arquitetura Interna do Componente O `SoftPromptGraphModel` é composto por dois componentes principais:

- **Um Codificador de Grafo (GNN):** A implementação de referência utiliza uma **Graph Attention Network v2 (GATv2)** [Brody, Alon e Yahav 2022], construída sobre a biblioteca **PyTorch Geometric (PyG)** [Fey e Lenssen 2019]. Sua função é processar o subgrafo refinado e gerar o *graph token*.
- **Um Modelo de Linguagem (LLM):** O componente contém uma referência a um LLM pré-treinado, carregado a partir da biblioteca **Hugging Face Transformers**. Para os experimentos, foram utilizados os modelos da família Llama (**Llama 3.1 8B**, **Llama 3.2 3B**) para analisar o equilíbrio entre performance e custo computacional.

A escolha da GATv2 se justifica por seu mecanismo de atenção, que permite ao modelo ponderar dinamicamente a importância dos nós e arestas no subgrafo com base em suas features, uma capacidade importante para lidar com os subgrafos de relevância variada gerados pelo pipeline. O processamento é otimizado para execução em GPU, embora a natureza compacta dos subgrafos refinados permita uma execução viável também em CPU.

Processo de Geração do *Graph Token* A transformação do subgrafo refinado em um *soft prompt* vetorial (o "graph token") segue o processo conceitual de 5 etapas definido na arquitetura. Na implementação, os *embeddings* de nós são inicializados com as representações pré-calculadas pelo modelo de *embeddings*, como por exemplo o `all-MiniLM-L6-v2`, garantindo consistência semântica. A GATv2 então aplica múltiplas camadas de propagação e atenção, seguidas por uma operação de *pooling* média (*mean pooling*) sobre todos os nós para gerar uma única representação vetorial para todo o subgrafo. Uma camada de projeção linear final alinha a dimensionalidade deste vetor com a do LLM.

Como dito anteriormente, um princípio fundamental do ARANDU é a modularidade. Embora a GATv2 seja a implementação padrão, o framework foi projetado para permitir a fácil substituição por outras variantes de GNNs, como **Graph Convolutional Networks (GCNs)** [Kipf e Welling 2017] ou **GraphSAGE** [Hamilton, Ying e Leskovec 2018]. Similarmente, a estratégia de integração do *graph token* com o LLM, implementada por padrão como concatenação contextual, é extensível para suportar abordagens alternativas como o **prefix tuning** [Li e Liang 2021]. Esta flexibilidade garante a adaptabilidade do ARANDU a diferentes modelos de linguagem e requisitos computacionais.

Ciclos de Treinamento e Inferência A implementação como um `nn.Module` unificado facilita o gerenciamento dos diferentes modos de operação. Uma decisão de design importante para manter o framework "leve" é que os pesos do LLM permanecem **congelados** (*frozen*) durante o treinamento. Na prática, ao entrar em modo de treino, o atributo `requires_grad=False` é configurado para todos os parâmetros do LLM. Dessa forma, o processo de otimização afeta apenas os pesos da GNN e de sua camada de projeção, ensinando o componente de grafo a gerar "soft prompts" que o LLM pré-treinado possa entender, sem o alto custo de ajustar o modelo de linguagem em si. Em modo de inferência, o pipeline completo é executado para gerar a resposta final.

Processo de Geração (Método `forward`) O método `forward` do módulo implementa a lógica de geração final. Ele recebe como entrada o subgrafo enriquecido e ponderado e a pergunta original. O processo ocorre em três etapas:

1. **Geração do Graph Token:** Primeiramente, o subgrafo de entrada é passado pela sub-arquitetura GATv2 para produzir o *graph token* vetorial.
2. **Construção do Prompt Híbrido:** Em seguida, um prompt textual é montado, combinando a pergunta original com a representação textual das triplas do subgrafo refinado. O *graph token* (um vetor) é então injetado no início da sequência de *embeddings* de entrada do LLM. Esta concatenação no nível dos *embeddings* é a implementação da estratégia de *soft prompting*.
3. **Geração da Resposta:** A sequência de *embeddings* híbrida (prefixo do grafo + texto) é passada através dos blocos *transformer* do LLM congelado para gerar a resposta final em linguagem natural.

Assim, durante a etapa de treinamento, o LLM possui um papel importante, ainda que não seja treinado. A partir da geração dele, é calculada a perda (*loss*) que o processo de treinamento objetiva minimizar. Dessa forma, o componente é capaz de otimizar os pesos da GNN e de sua camada de projeção, ensinando o componente de grafo a gerar "soft prompts" que o LLM pré-treinado possa entender, sem o alto custo de ajustar o modelo de linguagem em si. Em modo de inferência, o pipeline completo é executado para gerar a resposta final.

Saída do Componente Ao final da execução, o `SoftPromptGraphModel` retorna a resposta final em linguagem natural, gerada pelo LLM. Esta resposta é então encaminhada pelo `PipelineOrchestrator` para entrega ao usuário.

CAPÍTULO 6

Configuração Experimental

Este capítulo delineia a configuração experimental completa para a validação empírica do framework ARANDU proposto nesta tese. A avaliação experimental é estruturada de forma a responder sistematicamente às questões de pesquisa centrais, demonstrando tanto a eficácia quanto a eficiência da arquitetura neuro-simbólica desenvolvida.

A escolha dos *datasets*, *baselines* e métricas de avaliação foi orientada pela necessidade de estabelecer comparações justas e significativas com o estado da arte atual em RAG sobre grafos de conhecimento. Particular atenção foi dedicada à seleção de sistemas de comparação que representem diferentes paradigmas de abordagem ao problema. Estes incluem métodos puramente neurais, simbólicos e híbridos, permitindo assim uma análise abrangente do posicionamento do ARANDU no panorama atual de soluções.

6.1 Questões de Pesquisa

A validação empírica do framework ARANDU é orientada por um conjunto de questões de pesquisa formuladas para investigar, de maneira sistemática, a hipótese central desta tese. O objetivo é analisar crítica e quantitativamente as contribuições da arquitetura neuro-simbólica proposta em termos de desempenho, fidelidade e eficiência. Para tal, os experimentos foram estruturados para responder às seguintes questões:

QP1: Desempenho Geral O framework ARANDU demonstra desempenho competitivo ou superior às abordagens de estado da arte em RAG sobre grafos de conhecimento em métricas de recuperação de informação e acurácia de ponta a ponta? Especificamente, busca-se avaliar se a arquitetura neuro-simbólica proposta apresenta resultados melhores em relação a propostas já estabelecidas como NaiveRAG e GraphRAG, além de demonstrar competitividade em relação a sistemas do estado da arte, como o G-Retriever. Esta questão visa validar a eficácia geral da solução desenvolvida através de uma análise comparativa.

QP2: Otimização da Representação do Grafo Como o componente de orquestração lógica melhora a qualidade das evidências recuperadas e o desempenho final do sistema através da otimização da representação contextual? Esta questão investiga: (a) a metodologia de expansão lógica do subgrafo com regras mineradas offline para completar caminhos de raciocínio; (b) a estratégia de ponderação de relevância baseada na descoberta de caminhos de conexão via SPARQL entre "sementes"de evidências; e (c) o impacto da reordenação das triplas, guiada por esta ponderação, na redução do ruído contextual. O objetivo é quantificar como essa abordagem de otimização lógica melhora a fidelidade das respostas geradas e identifica as informações mais pertinentes para responder à pergunta formulada.

QP3: Operacionalização da Representação Estrutural via GNN e *Soft Prompting* Como a representação otimizada do subgrafo, operacionalizada através de *Graph Neural Networks*, afeta a geração da resposta pelo LLM? Esta questão investiga: (a) o efeito da injeção de informação estrutural via *graph tokens* comparado à linearização textual; (b) como a atribuição de pesos específicos às arestas influencia o processamento do modelo; (c) a capacidade dos *soft prompts* de capturar padrões relacionais que se perdem na serialização textual.

QP4: Eficiência Computacional O framework ARANDU mantém eficiência computacional compatível com o princípio de ser “leve”? Esta questão investiga: (a) a latência de inferência em comparação com outras abordagens (*baselines*); (b) o consumo de recursos computacionais (memória, GPU e CPU) durante a execução; (c) o custo computacional total do pipeline comparado com outras abordagens; (d) a escalabilidade do sistema em termos de tempo de resposta conforme o tamanho do grafo de conhecimento aumenta.

A resposta a estas quatro questões, através da análise quantitativa e qualitativa apresentada no Capítulo 7, forma a base da validação empírica desta tese.

6.2 Conjuntos de Dados (Datasets)

6.2.1 WebQSP

O *Web Questions Semantic Parses* (WebQSP) [Yih et al. 2016] é um *dataset* amplamente utilizado para avaliar sistemas de resposta a perguntas sobre grafos de conhecimento. Derivado do conjunto original *WebQuestions* [Berant et al. 2013], o WebQSP fornece anotações semânticas estruturadas que facilitam a avaliação de sistemas que operam sobre bases de conhecimento formalizadas.

O *dataset* é construído sobre o grafo de conhecimento Freebase [Bollacker et al. 2008], que contém informações factuais sobre entidades do mundo

real e suas relações. As perguntas no WebQSP são formuladas em linguagem natural e cobrem uma ampla variedade de domínios, incluindo geografia, entretenimento, esportes, política e ciências. Foi adotada a estratégia de pré-processamento apresentada em [Luo et al. 2024], utilizando um subgrafo de nós e arestas que se conectam em até 4 saltos (*4-hop*) com nós que aparecem na pergunta.

	question	answer	graph
0	what country is the grand bahama island in	[Bahamas]	[[Acklins, location.administrative_division.first_level_division_of, Bahamas], [Geographical Feature, freebase.type_profile.strict_included_types, Location], [Hurricane Edith, meteorology.tropical_cyclone.affected_areas, Bahamas], [Bahamas, meteorology.cyclone_affected_area.cyclones, 1933 Treasure Coast hurricane], [Bahamas, location.statistical_region.part_time_employment_percent, g.1hhc4bpw0], [m.04kc4ps, organization.organization_membership.member, Bahamas], [Bahamas, location.statistical_region.gender_balance_members_of_parliament, g.1hhc3s6q7], [Politician, freebase.type_profile.equivalent_topic, Politician], [Bahamas, location.statistical_region.gni_in_ppp_dollars, g.1245_mkq0], [Bahamas, location.statistical_region.co2_emissions_per_capita, g.12460kjms], [Tropical Storm Leslie, meteorology.tropical_cyclone.affected_areas, Bahamas], [Hurricane Flora, meteorology.tropical_cyclone.affected_areas, Bahamas], [Bahamas, base.aareas.schema.administrative_area.pertinent_type, Distric...
1	what kind of money to take to bahamas	[Bahamian dollar]	[[Hurricane Betsy, common.topic.notable_types, Tropical Cyclone], [United States of America, meteorology.cyclone_affected_area.cyclones, Hurricane Isabel], [Bahamas, location.statistical_region.co2_emissions_per_capita, g.12460kjms], [Hurricane Flora, meteorology.tropical_cyclone.affected_areas, Bahamas], [United States of America, location.country.first_level_divisions, New Jersey], [Cat Island, Bahamas, common.topic.notable_for, g.1255jz93k], [Tarpum Bay, common.topic.article, m.02vpd3x], [United States of America, base.locations.countries.states_provinces_within, North Carolina], [Dominica, meteorology.cyclone_affected_area.cyclones, Hurricane Hugo], [m.0khl4t6, olympics.olympic_athlete_affiliation.country, Bahamas], [Bahamas, location.statistical_region.renewable_freshwater_per_capita, g.1hhc3fsdd], [Sidney Poitier, freebase.valuenotation.is_reviewed, Country of nationality], [East Grand Bahama, common.topic.article, m.070vd9], [Haiti, meteorology.cyclone_affected_area.cyclones...
2	what character did john noble play in lord of the rings	[Denethor II]	[[John Noble, film.actor.film, m.0h1ldcx], [m.02_1x1x, base.gender.personal_gender_identity.gender_identity, Male], [m.0b4d5rz, award.award_nomination.award_nominee, Andy Serkis], [John Noble, people.person.place_of_birth, Port Pirie], [Sean Astin, people.person.profession, Actor], [John Noble by Gage Skidmore, common.image.size, m.0cmnjdl], [Fringe - Season 4, tv.tv_series_season.episodes, One Night in October], [John Noble, film.actor.film, m.0h5jyl5], [m.04kcrh0, tv.regular_tv_appearance.seasons, Fringe - Season 1], [Ian Holm, freebase.valuenotation.is_reviewed, Date of birth], [Andy Serkis, freebase.valuenotation.is_reviewed, Gender], [David Wenham, freebase.valuenotation.is_reviewed, Country of nationality], [Male, medicine.risk_factor.diseases, Transient ischemic attack], [Location, freebase.type_hints.included_types, Topic], [South Australia, base.aareas.schema.administrative_area.administrative_parent, Australia], [Actor, people.profession.specializations, Narrator], [Risen...

Figura 6.1: Exemplos de pergunta e resposta no WebQSP

A Figura 6.1 apresenta alguns exemplos de perguntas e respostas no WebQSP. Para cada pergunta, é apresentado o grafo de conhecimento que pode ser utilizado para responder à pergunta. Os grafos têm, em média, 4.252 triplas, podendo chegar a mais de 30.000 triplas em algumas perguntas.

O WebQSP compreende 4.700 pares pergunta-resposta, divididos em 2.826 exemplos de treinamento, 245 exemplos de validação e 1.629 exemplos de teste. A complexidade das perguntas varia desde consultas simples de um único salto (*single-hop*) até perguntas mais complexas que requerem múltiplos saltos através do grafo para alcançar a resposta correta. Esta diversidade na complexidade torna o WebQSP um *benchmark* robusto para avaliar tanto a capacidade de recuperação quanto o raciocínio dos sistemas de QA sobre grafos de conhecimento.

Uma característica importante do WebQSP é que as respostas são fornecidas como conjuntos de entidades, permitindo uma avaliação objetiva através de métricas baseadas em correspondência exata. A escolha deste *dataset* é motivada pela sua ampla adoção na comunidade de pesquisa e pela disponibilidade de resultados de *baseline* bem estabelecidos, proporcionando um contexto sólido para avaliar as contribuições do framework ARANDU.

6.2.2 MetaQA

O *MetaQA* [Zhang et al. 2018] é um *dataset* projetado para avaliar sistemas de resposta a perguntas que requerem raciocínio multi-salto sobre grafos de conhecimento. Diferentemente do WebQSP, que se baseia em um grafo de conhecimento de domínio aberto (Freebase), o MetaQA é construído sobre o grafo de conhecimento *WikiMovies*[Miller et al. 2016], que contém informações detalhadas sobre filmes, diretores, atores, gêneros e outros aspectos do domínio cinematográfico. O MetaQA é particularmente valioso para testar a capacidade de sistemas de QA em realizar inferências complexas através de múltiplas relações.

O *dataset* é estruturado hierarquicamente em três categorias baseadas na complexidade do raciocínio necessário: perguntas de 1-salto (9.947 exemplos), 2-saltos (14.872 exemplos) e 3-saltos (9.074 exemplos), totalizando aproximadamente 34 mil pares pergunta-resposta divididos em conjuntos de treinamento, validação e teste. Esta estrutura permite uma avaliação granular da capacidade de raciocínio do sistema em diferentes níveis de complexidade.

Todas as perguntas do dataset MetaQA utilizam o mesmo grafo de conhecimento, *WikiMovies*, contendo um total de 134.741 triplas, com 43.234 entidades únicas (nós) e 9 tipos de relações (arestas): *directed_by*, *written_by*, *starred_actors*, *release_year*, *in_language*, *has_tags*, *has_genre*, *has_imdb_votes*, *has_imdb_rating*.

As Tabelas 6.1, 6.2 e 6.3 apresentam a distribuição detalhada das perguntas por tipo e categoria no MetaQA. Como é possível observar através das tabelas, o MetaQA apresenta exemplos de diferentes níveis de complexidade, ilustrados a seguir:

Tabela 6.1: Distribuição de perguntas por tipo no MetaQA (1-hop)

Tipo de Pergunta	**Quantidade**
actor_to_movie	879
director_to_movie	553
movie_to_actor	1.105
movie_to_director	1.301
movie_to_genre	1.143
movie_to_language	294
movie_to_tags	846
movie_to_writer	1.091
movie_to_year	1.420
tag_to_movie	411
writer_to_movie	904

Tabela 6.2: Distribuição de perguntas por tipo no MetaQA (2-hops)

Tipo de Pergunta	**Quantidade**
actor_to_movie_to_actor	971
actor_to_movie_to_director	929
actor_to_movie_to_genre	823
actor_to_movie_to_language	326
actor_to_movie_to_writer	834
actor_to_movie_to_year	985
director_to_movie_to_actor	483
director_to_movie_to_director	164
director_to_movie_to_genre	533
director_to_movie_to_language	193
director_to_movie_to_writer	402
director_to_movie_to_year	594
movie_to_actor_to_movie	1.180
movie_to_director_to_movie	1.081
movie_to_writer_to_movie	896
writer_to_movie_to_actor	838
writer_to_movie_to_director	763
writer_to_movie_to_genre	844
writer_to_movie_to_language	250
writer_to_movie_to_writer	763
writer_to_movie_to_year	1.020

Tabela 6.3: Distribuição de perguntas por tipo no MetaQA (3-hops)

Tipo de Pergunta	**Quantidade**
movie_to_actor_to_movie_to_director	1.111
movie_to_actor_to_movie_to_genre	1.130
movie_to_actor_to_movie_to_language	851
movie_to_actor_to_movie_to_writer	1.172
movie_to_actor_to_movie_to_year	1.163
movie_to_director_to_movie_to_actor	1.110
movie_to_director_to_movie_to_genre	1.084
movie_to_director_to_movie_to_language	581
movie_to_director_to_movie_to_writer	1.054
movie_to_director_to_movie_to_year	1.155
movie_to_writer_to_movie_to_actor	812
movie_to_writer_to_movie_to_director	897
movie_to_writer_to_movie_to_genre	807
movie_to_writer_to_movie_to_language	398
movie_to_writer_to_movie_to_year	949

1-hop: *"Who directed the movie Forrest Gump?"* - Requer apenas um salto direto: `(Forrest Gump, directed_by, ?)`.

2-hops: *"What movies did the director of Forrest Gump direct?"* - O sistema deve primeiro identificar o diretor de "Forrest Gump" (salto 1: `(Forrest Gump, directed_by, Robert Zemeckis)`) e, em seguida, encontrar outros filmes dirigidos por essa pessoa (salto 2: `(?, directed_by, Robert Zemeckis)`).

3-hops: *"What genre are the movies directed by the director of Forrest Gump?"* - Exige três saltos sequenciais: primeiro identificar o diretor (salto 1: `(Forrest Gump, directed_by, Robert Zemeckis)`), depois encontrar os filmes dirigidos por ele (salto 2: `(?, directed_by, Robert Zemeckis)`), e finalmente determinar os gêneros desses filmes (salto 3: `(movies, has_genre, ?)`).

Uma característica distintiva do MetaQA é sua ênfase em perguntas que exigem raciocínio transitivo, como no exemplo acima. Isso torna o *dataset* uma ferramenta essencial para testar a eficácia do framework ARANDU em cenários que demandam encadeamento lógico e inferência multi-salto. A inclusão do MetaQA na avaliação experimental permite uma análise abrangente das capacidades do sistema em lidar com consultas complexas, validando a arquitetura neuro-simbólica proposta em contextos que vão além da recuperação simples de fatos.

6.3 Sistemas de Base (*Baselines*)

6.3.1 NaiveRAG

Para avaliar a contribuição dos componentes propostos na arquitetura do framework ARANDU, o primeiro sistema de base é o *NaiveRAG*. Ele representa uma implementação padrão do paradigma RAG baseado puramente em similaridade semântica, tratando os dados como texto simples independentemente de sua estrutura original. A comparação direta com este modelo permite isolar e medir o impacto específico das inovações propostas, estabelecendo uma linha de base que utiliza apenas recuperação por *embedding* e linearização textual das evidências.

O NaiveRAG representa a hipótese nula de que a recuperação de triplas baseada unicamente em similaridade semântica é suficiente para gerar respostas corretas. Ao comparar o desempenho do ARANDU com o do NaiveRAG, os experimentos buscam demonstrar a necessidade de mecanismos de contextualização mais avançados, que considerem a estrutura do grafo e a lógica das relações, indo além da simples correspondência de *embeddings*.

Na configuração experimental, o pipeline do NaiveRAG é instanciado da seguinte forma:

- **Recuperação:** Cada tripla (sujeito, predicado, objeto) do grafo de conhecimento é convertida para texto e codificada em um *embedding* denso utilizando o modelo Sentence-Transformers (e.g. `all-MiniLM-L6-v2`). Dada uma pergunta, seu *embedding* é gerado com o mesmo modelo, e a similaridade de cosseno é usada para ordenar as triplas da base de conhecimento. As top-K triplas mais similares são selecionadas como evidência. A fim de realizar uma comparação justa, e com menos pontos de mudança, os modelos de *embeddings* utilizados foram os mesmos do ARANDU. Adicionalmente, o valor de K utilizado foi a média de triplas recuperadas pelo ARANDU para cada dataset (K = 24 para WebQSP, K = 10 para MetaQA - 1-hop e K = 18 para MetaQA - 2-hop).
- **Aumento:** As K triplas recuperadas são linearizadas em texto simples, seguindo o formato "sujeito predicado objeto"(e.g., "Barack Obama *place of birth* Hawaii"). As triplas são concatenadas sequencialmente em um único bloco de texto, preservando a ordem do ranking de similaridade obtido na fase de recuperação. Este processo de linearização representa uma transformação determinística de formato (grafo $\rightarrow$ texto). Nele é realizada a conversão da estrutura relacional em contexto textual compatível com o modelo de linguagem, sem qualquer processamento adicional ou consideração sobre as conexões estruturais entre as triplas.

- **Geração:** O contexto textualizado é inserido em um prompt, junto com a pergunta original, e submetido ao modelo de linguagem para a geração da resposta final.

A limitação fundamental do NaiveRAG, que serve de base para a nossa comparação, é sua vulnerabilidade a ruído. O sistema pode recuperar fatos que são semanticamente similares à pergunta, mas irrelevantes ou incorretos no contexto da consulta. A ausência de mecanismos de otimização da representação faz do NaiveRAG um controle experimental para demonstrar o valor dos componentes de raciocínio propostos no ARANDU.

6.3.2 GraphRAG

Como visto na Seção 3.1, o GraphRAG representa uma evolução fundamental em relação ao NaiveRAG, incorporando o contexto estrutural do grafo de conhecimento diretamente na fase de recuperação. Diferentemente de abordagens que tratam a base de conhecimento como uma coleção de documentos textuais isolados, o GraphRAG recupera subgrafos contextuais, preservando as relações explícitas entre as entidades. Esta abordagem visa mitigar os problemas de ruído e falta de coerência observados em sistemas RAG tradicionais. A inclusão do GraphRAG como sistema de base é essencial para avaliar como o framework ARANDU se posiciona em relação a métodos que já exploram a topologia do grafo.

O GraphRAG aborda a limitação do RAG clássico em responder perguntas que requerem raciocínio sobre múltiplas entidades conectadas. Em vez de operar com fragmentos de texto isolados, o GraphRAG primeiro identifica entidades relevantes e, em seguida, explora suas conexões para construir um subgrafo coeso que serve como evidência.

Na configuração experimental, o pipeline do GraphRAG é instanciado da seguinte forma:

- **Recuperação de Entidades Iniciais:** Dada uma pergunta, uma busca por similaridade semântica (recuperação densa) é executada sobre um índice de todas as entidades do grafo. Esta etapa identifica um conjunto de nós-âncora (anchor nodes) que são semanticamente relevantes para a consulta.
- **Expansão e Construção do Subgrafo:** A partir dos nós-âncora, um algoritmo de travessia ou expansão de grafo é aplicado para construir um subgrafo conectado. Métodos comuns para esta etapa incluem a expansão por vizinhança (k-hop), beam search, ou algoritmos de otimização como o *Prize-Collecting Steiner Tree (PCST)*. O objetivo é extrair uma estrutura que contenha os nós-âncora e os caminhos mais informativos que os conectam.

- **Aumento (Linearização do Contexto):** O subgrafo resultante, com suas entidades e relações, é serializado para um formato textual. Essa "linearização"transforma a estrutura do grafo em uma sequência de sentenças ou triplas (e.g., "Barack Obama, nasceu em, Honolulu"), que pode ser processada por um modelo de linguagem.
- **Geração:** O texto que representa o subgrafo é concatenado com a pergunta original e fornecido como contexto para o LLM, que então gera a resposta final em linguagem natural.

Embora o GraphRAG seja um avanço significativo, ele possui limitações que motivam a necessidade de abordagens mais sofisticadas. Primeiramente, a qualidade do subgrafo é altamente dependente da precisão da etapa inicial de recuperação de entidades; erros nessa fase se propagam e podem resultar em um contexto equivocado. Em segundo lugar, a etapa de expansão, embora estruturalmente consciente, carece de mecanismos de validação lógica, podendo incluir caminhos factualmente corretos mas contextualmente irrelevantes ou ilógicos. Finalmente, o processo de linearização para texto pode resultar na perda de informações topológicas complexas, que não são facilmente capturadas em formato sequencial.

A comparação entre o ARANDU e o GraphRAG é, portanto, relevante. Ela permite demonstrar como nossa abordagem neuro-simbólica, com seu componente de orquestração lógica explícito e o uso de GNNs para preservar a estrutura do grafo, supera essas limitações, resultando em maior fidelidade e precisão.

6.3.3 G-Retriever

Como visto na Seção 3.1.4, o G-Retriever [He et al. 2024] representa um marco na evolução dos sistemas RAG para grafos de conhecimento ao introduzir otimização de subgrafos baseada em algoritmos de árvore de Steiner. Esta abordagem fundamenta-se na premissa de que a recuperação eficaz deve identificar subgrafos conexos e informativos. Esses subgrafos capturam caminhos de raciocínio multi-hop entre entidades relevantes à pergunta formulada. A inclusão do G-Retriever como sistema de base é importante para avaliar como o framework ARANDU se posiciona em relação a métodos que já exploram otimização estrutural avançada.

Na configuração experimental, o pipeline do G-Retriever é instanciado da seguinte forma:

- **Identificação de Entidades:** O sistema utiliza reconhecimento de entidades nomeadas e *entity linking* para mapear menções na pergunta às entidades correspondentes no grafo de conhecimento. Esta fase estabelece os nós âncora que servirão como pontos de partida para a construção do subgrafo.

- **Otimização de Subgrafo:** A recuperação é formulada como um problema de Prize-Collecting Steiner Tree (PCST). Nós recebem prêmios baseados em sua relevância à pergunta, enquanto arestas possuem custos associados. O algoritmo busca identificar o subgrafo de menor custo que conecta entidades relevantes e maximiza a informação capturada.
- **Contextualização via GNN:** O subgrafo otimizado é processado por uma Rede Neural de Grafos (GNN) que gera representações contextualizadas. Estas preservam tanto informações semânticas locais quanto padrões estruturais globais do subgrafo recuperado.
- **Geração:** As representações contextualizadas são integradas ao prompt do modelo de linguagem através de técnicas de *soft prompting*. Isso permite que informações estruturais influenciem diretamente os mecanismos de atenção durante a geração da resposta.

O G-Retriever introduz otimização estrutural via PCST e integração de GNNs para preservar informação topológica. Contudo, opera exclusivamente sobre conectividade topológica sem incorporar descoberta de caminhos lógicos. A ausência de orquestração lógica pode resultar em caminhos estruturalmente coesos, mas logicamente incompletos.

A comparação com ARANDU demonstra como a arquitetura neuro-simbólica estende a otimização estrutural. Esta extensão incorpora descoberta de caminhos lógicos e enriquecimento via regras mineradas. A análise empírica avalia se estes componentes oferecem ganhos sobre a otimização puramente estrutural.

6.4 Detalhes de Implementação e Hiperparâmetros

Esta seção detalha as configurações específicas dos componentes neurais do *framework* ARANDU, bem como os hiperparâmetros empregados durante a fase experimental. A apresentação minuciosa desses detalhes, incluindo o ambiente computacional, é essencial para assegurar a reprodutibilidade dos resultados obtidos. Todos os experimentos foram conduzidos em uma estação de trabalho equipada com uma GPU NVIDIA RTX 4070 de 12 GB de VRAM e um processador Intel Core i5-14600K com 20 *threads*.

6.4.1 Configuração do Large Language Model (LLM)

Dentre os aspectos relevantes da implementação, destaca-se o uso de técnicas de quantização para compressão dos modelos de linguagem (LLM). A quantização consiste em representar os parâmetros (pesos e ativações) utilizando menos bits do que o padrão de ponto flutuante (tipicamente FP32 ou FP16). Esse processo tem como objetivo reduzir o

consumo de memória e acelerar a inferência. Trata-se de uma abordagem que viabiliza a execução de modelos de grande porte em hardware com recursos limitados, como GPUs de menor capacidade ou mesmo CPUs. Em vez de armazenar cada valor com alta precisão, a quantização mapeia os valores contínuos para um conjunto discreto de níveis, geralmente utilizando representações em 8, 4 ou até 2 bits. Embora essa aproximação possa introduzir pequenas perdas de precisão numérica, avanços recentes em algoritmos de quantização permitem preservar a maior parte do desempenho original do modelo. Dessa forma, a quantização tornou-se uma estratégia fundamental para viabilizar aplicações práticas de modelos de linguagem de larga escala em ambientes restritos.

A eficácia dessa abordagem é evidenciada por técnicas modernas, como quantização pós-treinamento [NVIDIA 2023] e técnicas adaptativas (por exemplo, NF4) [Dettmers et al. 2023]. Benchmarks de referência, como o MMLU[1], mostram que tais técnicas preservam entre 95% e 97% da acurácia de modelos em FP16. Isso viabiliza a execução de grandes modelos em GPUs com restrições de memória. Além disso, o debate sobre a precisão de diferentes tipos de quantização é relevante; embora a quantização em INT8 seja tradicionalmente vista como mais estável em cenários de contexto longo [Shen et al. 2025], evidências mostram que modelos maiores em INT4 superam modelos menores em INT8. Isso ocorre porque o maior número de parâmetros compensa a perda de precisão numérica introduzida pela quantização mais agressiva.

No presente trabalho, optamos por empregar o *Llama 3.2 3B* em INT8 e o *Llama 3.1 8B* em INT4, ambos congelados, apenas adicionando tokens de *soft prompt* via GNN. Dessa forma, a quantização não exerce impacto relevante no resultado final. Mas, garante a execução eficiente em ambientes com recursos limitados, como GPUs domésticas, com até 12 GB de VRAM. Dessa forma, concilia-se custo computacional reduzido com a preservação da capacidade representacional dos modelos.

A seguir, detalhamos os parâmetros de configuração e execução.

- **Modelos Base:** Llama 3.1 8B Instruct e Llama 3.2 3B Instruct.
- **Quantização:** A fim de otimizar o uso de recursos computacionais, os modelos foram quantizados com o auxílio da biblioteca *bitsandbytes* [Dettmers et al. 2023]. O modelo Llama 3.1 8B Instruct foi quantizado para 4-bits (NF4) [Dettmers et al. 2023], enquanto o Llama 3.2 3B Instruct foi submetido a uma quantização de 8-bits.
- **Framework:** O modelo foi carregado e executado utilizando a biblioteca *Transformers* [Wolf et al. 2020] com PyTorch [Paszke et al. 2019].

[1] O MMLU (*Massive Multitask Language Understanding*) é um *benchmark* amplamente utilizado para avaliar o conhecimento e raciocínio de modelos de linguagem em 57 disciplinas acadêmicas.

- **Implementação da Atenção:** Foi utilizada a implementação `flash_attention_2` [Dao 2023] para otimizar a velocidade e o consumo de memória do mecanismo de atenção.
- **Tamanho Máximo de Tokens de Entrada (texto):** O contexto textual de entrada para o LLM foi limitado a 512 tokens.
- **Tamanho Máximo de Tokens de Saída:** A geração de respostas foi configurada para um máximo de 512 tokens.

Construção do *Prompt* com *Chat Template*

A formatação do *prompt* de entrada é um aspecto fundamental para garantir que o LLM processe a informação de maneira eficaz. Arquiteturas como o G-Retriever [He et al. 2024] adotam uma abordagem de construção manual, onde *tokens* especiais são concatenados diretamente ao texto para indicar o início e o fim de diferentes segmentos, como o contexto do grafo e a pergunta do usuário. Essa técnica, embora funcional, exige um controle preciso da concatenação de *strings* e *tokens*, o que pode ser propenso a erros e específico para cada modelo.

No presente trabalho, optou-se por uma abordagem mais moderna e robusta, utilizando o recurso de *chat template* disponibilizado pela biblioteca *Transformers*. Em vez de construir o *prompt* manualmente, a entrada é estruturada como uma conversa com papéis distintos (como `user` e `assistant`). O conjunto de papéis foi estendido para incluir um papel customizado, `graph`, dedicado a encapsular a informação contextual proveniente do grafo de conhecimento.

O *tokenizer* do modelo aplica um gabarito predefinido que insere automaticamente todos os *tokens* especiais necessários. Cada família de modelos adota um padrão específico (como, por exemplo, Llama 3 adota `<|begin_of_text|>`, `<|start_header_id|>`, `<|end_header_id|>`, `<|eot_id|>`). Esses *tokens* são definidos durante o treinamento do modelo, sendo abstraídos pelo *tokenizer*. O recurso de *chat template* insere automaticamente esses *tokens* e formata a entrada de acordo com o padrão em que o modelo foi treinado.

Essa metodologia oferece vantagens de robustez e manutenibilidade. Para integrar a representação do grafo de forma estruturada, foi introduzido o papel `graph`. A conversa é iniciada com este papel, contendo um *token* de espaço reservado (`<GRAPH_TOKEN>`). Em tempo de execução, o *embedding* vetorial gerado pela GNN substitui programaticamente o *embedding* deste *token*, inserindo a representação neural do grafo diretamente no fluxo de entrada do LLM, antes da camada de atenção. Essa técnica preserva a estrutura do *chat template* e permite a injeção de informação multimodal (textual e gráfica) de maneira coesa.

6.4.2 Componentes de Recuperação e Orquestração Lógica

A construção do subgrafo de contexto depende de um conjunto de componentes e hiperparâmetros que definem a qualidade e a relevância da informação recuperada.

Mineração de Regras Lógicas

As regras lógicas utilizadas para expandir o subgrafo inicial foram mineradas a partir do grafo de conhecimento completo utilizando o sistema AMIE3 [Lajus, Galárraga e Suchanek 2020]. Os parâmetros de mineração foram definidos para extrair regras com um grau mínimo de generalidade e confiança, conforme detalhado abaixo:

- **Suporte Mínimo:** Para o conjunto WebQSP, que fornece um grafo específico para cada questão, foi adotado um valor dinâmico de suporte mínimo. Esse valor corresponde ao inteiro mais próximo de 0,25% do número total de triplas do grafo da questão. Para o conjunto MetaQA, que utiliza um único grafo para todas as perguntas, foram realizados testes com valores entre 5 e 50. Definiu-se o valor 10, o que resultou em aproximadamente 2 mil regras extraídas para esse grafo.
- **Confiança PCA Mínima:**[2] Foi definido o valor de 0,8. Este valor foi selecionado após experimentação com valores no intervalo [0,6, 0,9]. O valor 0,8 apresentou o melhor equilíbrio entre a quantidade de regras extraídas e sua qualidade. Valores inferiores a 0,8 geraram excesso de regras triviais ou ruidosas. Valores superiores produziram um conjunto muito restrito de regras, limitando a expansão do subgrafo.

Recuperação Semântica

A recuperação de nós e arestas relevantes para a pergunta do usuário foi realizada por meio de busca vetorial em um índice semântico.

- **Modelos de Embedding:** Os modelos utilizados para gerar as representações vetoriais de nós e relações foram o `all-MiniLM-L6-v2`, para o conjunto WebQSP, e o `all-mpnet-base-v2`, para o conjunto MetaQA. Tais modelos são disponibilizados pela biblioteca *Sentence-Transformers* [Reimers e Gurevych 2019].
- **Top-K Entidades:** Foram recuperadas as 3 entidades mais semanticamente similares à pergunta.

[2] A Confiança PCA (do inglês, *Partial Completeness Assumption*) é uma métrica de confiança de regras proposta pelo sistema AMIE. Diferente da confiança padrão, que exige uma consulta custosa sobre todo o grafo de conhecimento para sua computação, a Confiança PCA é uma aproximação que assume que qualquer tripla não presente no grafo é falsa. Essa suposição permite uma estimativa de confiança muito mais eficiente. Assim, viabiliza a mineração de regras em grafos de grande escala [Lajus, Galárraga e Suchanek 2020].

- **Top-K Relações:** Foram recuperadas as 5 relações (triplas) mais semanticamente similares à pergunta.

Como apresentado no Capítulo 5, após a recuperação das entidades e relações candidatas, emprega-se o algoritmo *Prize-Collecting Steiner Tree* (PCST), com uma implementação baseada na abordagem do G-Retriever [He et al. 2024]. Para o PCST, foram utilizados os seguintes parâmetros:

- **Definição dos Prêmios (Nós e Arestas):** Diferente de usar diretamente a pontuação de similaridade, os prêmios foram definidos com base no *ranking* discreto dos elementos recuperados. Para os nós, as Top-K entidades recebem prêmios inteiros decrescentes (ex: para k=3, os prêmios são 3.0, 2.0 e 1.0). Uma lógica similar é aplicada às arestas, que também recebem prêmios com base em seu *ranking* de similaridade.
- **Definição dos Custos (Arestas):** O custo de cada aresta é calculado dinamicamente com base em seu prêmio. O custo final é dado por `Custo Base - Prêmio da Aresta`, onde o custo base é ajustado para garantir que pelo menos uma aresta seja selecionada. Especificamente, o custo base é limitado pelo valor `Prêmio Máximo × (1 - c/2)`, onde `c` é um parâmetro de controle. Dessa forma, arestas com maior relevância têm custo menor, aumentando sua probabilidade de inclusão no subgrafo final.
- **Virtualização de Arestas de Alta Relevância:** Quando o prêmio de uma aresta excede o custo base, aplica-se uma técnica de virtualização. A aresta original é substituída por um nó virtual conectado aos nós originais por duas novas arestas de custo zero. O prêmio excedente (`Prêmio da Aresta - Custo Base`) é atribuído ao nó virtual. Essa abordagem garante que arestas altamente relevantes sejam preservadas na solução final, mesmo quando seus prêmios excedem os custos permitidos pelo algoritmo PCST.
- **Parâmetros de Controle:**
  - **Nó Raiz:** A configuração utilizada é de um grafo não enraizado (*unrooted*).
  - **Método de Poda (*Pruning*):** Foi utilizado o algoritmo de Goemans-Williamson [Goemans e Williamson 1995][3].

[3] O algoritmo de Goemans-Williamson é uma técnica de aproximação para problemas de florestas com restrições que oferece garantias teóricas de aproximação. No contexto do PCST, este algoritmo remove iterativamente componentes de baixo valor da árvore, otimizando a relação entre prêmios coletados e custos incorridos.

6.4.3 Configuração da *Graph Neural Network* (GNN)

Como dito no Capítulo 5, o componente de representação neural do grafo utiliza uma *Graph Attention Network v2* (GATv2) [Brody, Alon e Yahav 2022] para codificar o subgrafo recuperado em uma representação vetorial. Os principais hiperparâmetros da arquitetura são:

- **Arquitetura:** Graph Attention Network v2 (GATv2).
- **Número de Camadas:** 4 camadas de GATv2.
- **Número de Heads de Atenção:** 4 *heads* por camada.
- **Dimensão de Entrada:** Vetores de características dos nós com 384 ou 768 dimensões, dependendo do modelo de *embedding* utilizado.
- **Tamanho da Camada Oculta:** Dimensão de 1024 para as camadas intermediárias.
- **Tamanho da Camada de Saída:** A representação de saída da GNN para cada nó possui dimensão de 1024.
- **Função de Ativação:** `LeakyReLU`[4], conforme a implementação padrão da GATv2.
- **Projetor:** A representação agregada do grafo (via *mean pooling*) é processada por um Multi-Layer Perceptron (MLP) de duas camadas com função de ativação `Sigmoid`, que projeta o vetor do grafo para o espaço de *embeddings* do LLM (3072 dimensões para o *Llama 3.2 3B Instruct* e 4096 dimensões para o *Llama 3.1 8B Instruct*).

A escolha destes hiperparâmetros foi guiada por uma fase de experimentação preliminar, buscando um balanço entre capacidade de representação e eficiência computacional.

6.4.4 Treinamento do Módulo GNN

O treinamento do módulo GNN, que ajusta os pesos do projetor para alinhar as representações do grafo com o espaço de *embeddings* do LLM, foi conduzido com os seguintes hiperparâmetros:

- **Otimizador:** AdamW [Loshchilov e Hutter 2019].
- **Taxa de Aprendizagem:** 1e-5.
- **Decaimento de Peso (Weight Decay):** 0,05.
- **Tamanho do Lote (Batch Size):** Entre 4 e 8 para o *Llama 3.2 3B Instruct* e entre 2 e 4 para o *Llama 3.1 8B Instruct*.

[4] A *Leaky Rectified Linear Unit* é uma variação da função ReLU que permite um pequeno gradiente para entradas negativas, evitando o problema dos neurônios inativos (*dying ReLUs*). A implementação padrão do GATv2 utiliza um coeficiente de inclinação de 0.2 para valores negativos.

- **Número de Épocas:** 10, com paciência[5] de 2 para *early stopping*.

6.5 Métricas e Protocolo de Avaliação

A validação empírica do framework ARANDU adota um protocolo de avaliação estruturado em duas fases: (I) Avaliação da Recuperação e (II) Avaliação da Resposta Final. A separação em duas etapas oferece benefícios metodológicos específicos e alinha-se com a arquitetura modular do ARANDU.

A avaliação da recuperação foca nas fases I e II, compreendendo os módulos de recuperação textual e estruturada, e o módulo de orquestração lógica. Com isso, avalia a qualidade das evidências recuperadas antes de serem processadas pelo LLM. A avaliação da resposta final examina o pipeline como um todo. Inclui a representação neural via GNN e *soft prompting* e a geração realizada pela LLM. Assim, avalia como a injeção de informação estrutural via *graph tokens* impacta a capacidade do sistema de sintetizar respostas corretas.

Esta divisão permite isolar e quantificar as contribuições específicas da otimização estrutural do grafo versus sua operacionalização neural. O que facilita a identificação de limitações em componentes específicos do pipeline. O protocolo adotado segue padrões estabelecidos na literatura de QA sobre grafos de conhecimento[Gao et al. 2023], adaptado para contemplar as inovações neuro-simbólicas propostas.

As métricas listadas a seguir foram utilizadas tanto para avaliar a etapa de recuperação quanto para avaliar a etapa de geração. Tais métricas são comuns na literatura de QA sobre grafos de conhecimento e garantem uma boa avaliação da cobertura e precisão tanto das evidências recuperadas quanto das respostas geradas.

Recall@K Mede a proporção de entidades-resposta corretas presentes entre os K itens melhor ranqueados pelo sistema de recuperação. Esta métrica avalia a capacidade do sistema de capturar as evidências necessárias para responder corretamente à pergunta, independentemente de sua posição no ranking. Formalmente, é definida como:

$$\text{Recall@K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{|\text{Respostas Corretas}_i \cap \text{Top-K Recuperados}_i|}{|\text{Respostas Corretas}_i|} \tag{6-1}$$

Hits@K Calcula a proporção de perguntas para as quais pelo menos uma resposta correta está presente entre os K itens melhor ranqueados. Esta métrica avalia a eficácia

[5] A paciência define o número de épocas consecutivas sem melhoria na métrica de validação antes da interrupção automática do treinamento.

binária do sistema de recuperação, indicando se o sistema consegue identificar ao menos uma evidência relevante dentro do conjunto de candidatos selecionados. É definida formalmente como:

$$\text{Hits@K} = \frac{1}{|Q|}\sum_{i=1}^{|Q|} 1(|\text{Respostas Corretas}_i \cap \text{Top-K Recuperados}_i| > 0) \tag{6-2}$$

onde $1(\cdot)$ é a função indicadora que retorna 1 se a condição é verdadeira e 0 caso contrário.

Precision@K Quantifica a proporção de entidades na resposta gerada que estão corretas segundo o gabarito. Esta métrica avalia a precisão do sistema, penalizando respostas que incluem entidades irrelevantes ou incorretas. É definida como:

$$\text{Precision@K} = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{|\text{Respostas Corretas}_i \cap \text{Top-K Recuperados}_i|}{|\text{Top-K Recuperados}_i|} \tag{6-3}$$

Além das métricas descritas acima, na etapa de avaliação da recuperação, foram utilizadas métricas focadas na posição da evidência recuperada no ranking. Estas métricas são fundamentais para avaliar não apenas se as evidências corretas foram identificadas, mas também quão bem o sistema as prioriza em relação a informações menos relevantes. A qualidade do ranqueamento é particularmente importante em sistemas de RAG, onde o LLM processa sequencialmente as evidências fornecidas, sendo mais influenciado por aquelas posicionadas no início do contexto[Sun et al. 2024].

Mean Reciprocal Rank (MRR) Calcula a média das posições recíprocas da primeira resposta correta encontrada no ranking. Esta métrica penaliza sistemas que posicionam respostas corretas em posições mais baixas, favorecendo aqueles que recuperam evidências relevantes nas primeiras posições. É calculada como:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \tag{6-4}$$

onde rank_i é a posição da primeira resposta correta para a pergunta i.

Normalized Discounted Cumulative Gain (nDCG@K) Quantifica a qualidade do ranking considerando tanto a relevância quanto a posição dos itens recuperados. Esta métrica atribui maior peso às respostas corretas posicionadas no topo da lista, utilizando um fator de desconto logarítmico. O nDCG@K é normalizado pelo DCG ideal, permitindo comparações entre diferentes consultas:

$$\text{nDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}} \tag{6-5}$$

onde DCG@K (Discounted Cumulative Gain) é calculado como:

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{6-6}$$

sendo rel_i a relevância binária do item na posição i (1 se correto, 0 caso contrário). O IDCG@K (Ideal DCG) representa o valor máximo de DCG@K obtido ao ordenar idealmente os itens por relevância, servindo como fator de normalização para garantir que nDCG@K esteja no intervalo [0,1].

Estas métricas são complementares e oferecem uma visão abrangente do desempenho da recuperação. O MRR prioriza a precisão posicional, e o nDCG@K considera simultaneamente relevância e ordenação. Para todos os experimentos conduzidos, foi adotado o valor $K = 10$, seguindo o protocolo padrão estabelecido na literatura de QA sobre grafos de conhecimento [Gao et al. 2023]. Na etapa de avaliação da recuperação, também foi utilizado o valor $K = 20$ para Recall e Hits, para uma análise mais abrangente da cobertura da recuperação.

Na avaliação da resposta final gerada pelo framework ARANDU também foi utilizada a métrica F1-Score, como descrito a seguir.

F1-Score Calcula a média harmônica entre precisão (Precision) e cobertura (Recall) das entidades na resposta, oferecendo uma medida equilibrada que considera simultaneamente a capacidade de identificar entidades corretas e evitar falsas inclusões. É computado como:

$$\text{F1-Score} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{6-7}$$

onde Precision_i e Recall_i são calculados individualmente para cada pergunta i.

Estas métricas operam de forma complementar para prover uma avaliação multidimensional da qualidade das respostas. O Hits@10, no contexto da geração, estabelece um critério abrangente de eficácia ao verificar se pelo menos uma resposta correta está presente na resposta final. Esta métrica é importante para avaliar a capacidade básica do sistema de identificar respostas relevantes. O Recall@10 quantifica a capacidade de cobertura completa das entidades relevantes. Mede a proporção de respostas corretas que o sistema consegue gerar dentre todas as possíveis. Na etapa de geração, é comum que o Recall@10 seja maior que aquele aferido na etapa de recuperação. Esse efeito se dá pela combinação do conhecimento recuperado com já presente no LLM, usado durante seu treinamento.

O Precision@10 avalia a minimização de inclusões incorretas. Quantifica a proporção de entidades corretas dentre as geradas pelo sistema. Na etapa de geração, essa métrica é particuparmente importante para indicar niveis altos de alucinação ou verbosidade. O F1-Score sintetiza os aspectos de precisão e cobertura em uma métrica unificada. Equilibra sensibilidade e especificidade, proporcionando uma base objetiva para comparações inter-sistêmicas. A aplicação combinada deste conjunto de métricas possibilita a identificação de características específicas. Inclui tendências de sub-cobertura, sobregeração ou desequilíbrios entre precisão e abrangência.

CAPÍTULO 7

Análise Experimental e Discussão

Este capítulo apresenta a análise experimental e a discussão dos resultados obtidos com o *framework* ARANDU. A estrutura da análise foi projetada para responder, de forma sistemática, às questões de pesquisa formuladas no Capítulo 6, validando a hipótese central desta tese. Cada seção a seguir aborda uma questão específica, combinando métricas quantitativas e análises qualitativas para avaliar o desempenho, a fidelidade e a eficiência da arquitetura neuro-simbólica proposta.

A Seção 7.1 aborda a **QP1**, apresentando uma análise de desempenho geral e comparativo do ARANDU frente a *baselines* do estado da arte. Em seguida, a Seção 7.2 aprofunda-se na análise isolada da etapa de recuperação. Esta análise detalha os ganhos do ARANDU na etapa de recuperação, focando na **QP2**. Assim, investigando como o componente de orquestração lógica otimiza a representação do grafo e melhora a qualidade das evidências. A Seção 7.3 foca na **QP3**, analisando a operacionalização da representação estrutural via GNN e seu impacto na modulação dos mecanismos de atenção do LLM. Por fim, a Seção 7.4 responde à **QP4**, com uma análise da eficiência computacional que valida o princípio "leve" do *framework*. O capítulo encerra com estudos de caso, apresentado na Seção 7.5, que ilustram a sinergia dos componentes.

7.1 Análise de Desempenho Geral e Comparativo

A análise de desempenho geral e comparativo do *framework* ARANDU é importante para validar sua eficácia em relação aos *baselines* do estado da arte. Esta seção foca em responder à **QP1**, que investiga como o ARANDU se posiciona em termos de precisão, revocação e outras métricas relevantes quando comparado a modelos estabelecidos. A avaliação é conduzida utilizando os conjuntos de dados WebQSP e MetaQA, ambos amplamente reconhecidos na área de Resposta a Perguntas sobre Grafos de Conhecimento e detalhados nas Seções 6.2.1 e 6.2.2, respectivamente.

Para garantir uma análise robusta, foram selecionados dois modelos de linguagem de última geração, versões de 3 bilhões e 8 bilhões de parâmetros do Llama, que servem como base para a comparação. Cada modelo foi configurado sob condições con-

troladas, assegurando que as diferenças de desempenho possam ser atribuídas às características intrínsecas de cada abordagem. No entanto, é importante notar que ainda podem ocorrer variações devido à natureza não determinística dos modelos generativos, como os LLMs, o que pode influenciar na reprodutibilidade dos resultados observados.

7.1.1 Resultados no Dataset WebQSP

A avaliação de desempenho geral considera o processo completo, desde a recuperação até a geração. O foco é a qualidade das respostas finais produzidas pelo ARANDU após o processamento completo do *pipeline*. Esta análise é conduzida utilizando métricas determinísticas baseadas em exatidão, que comparam diretamente as entidades presentes na resposta gerada com o conjunto de respostas de referência (*ground-truth*). As principais métricas empregadas são: **Recall@10**, que avalia a capacidade do sistema de recuperar respostas corretas entre os dez primeiros resultados; **Hits@10**, que mede a proporção de perguntas com pelo menos uma resposta correta nos dez primeiros resultados; **Precision@10**, que quantifica a precisão das respostas nos dez primeiros resultados; e **F1-Score**, que calcula a média harmônica entre precisão e revocação das entidades identificadas. Essas métricas são fundamentais para avaliar tanto a capacidade de recuperação quanto a qualidade das respostas geradas, proporcionando uma visão abrangente do desempenho do sistema em diferentes aspectos da tarefa de QA sobre grafos de conhecimento.

A integração dos módulos de orquestração lógica e representação neural via GNN demonstra impacto positivo na qualidade das respostas geradas. O módulo de orquestração lógica desprioriza caminhos irrelevantes ou inconsistentes do subgrafo recuperado, reduzindo o ruído contextual apresentado ao modelo de linguagem. Simultaneamente, o módulo de representação neural via GNN transforma o subgrafo refinado em representações contínuas que preservam a estrutura relacional, facilitando a interpretação pelo LLM. Esta abordagem resulta em respostas mais precisas e fiéis ao conhecimento estruturado, evidenciando a eficácia da arquitetura neuro-simbólica proposta.

A Tabela 7.1 apresenta os resultados comparativos no conjunto WebQSP utilizando o modelo Llama 3.2 3B. Os resultados demonstram que o ARANDU supera consistentemente os métodos de referência em todas as métricas avaliadas. Especificamente, o ARANDU alcança um Recall@10 de 54,91%, superando o G-Retriever (50,76%) e o GraphRAG (36,84%). Na métrica Hits@10, o ARANDU atinge 70,62%, comparado aos 66,81% do G-Retriever e 55,07% do GraphRAG. O que demonstra que o ARANDU consegue trazer mais respostas corretas entre as primeiras. Em relação à Precision@10, o ARANDU obtém 36,59%, superando tanto o GraphRAG (35,87%) quanto o G-Retriever (4,08%). Esta métrica indica que o ARANDU não apenas recupera mais respostas corre-

tas, mas também mantém maior precisão na ordenação dos resultados, reduzindo a presença de respostas incorretas entre as primeiras posições. Já o F1-Score mostra uma diferença ainda mais pronunciada, com o ARANDU obtendo 43,91%, significativamente superior aos 20,00% do G-Retriever e 26,96% do GraphRAG. Estes resultados indicam que a arquitetura neuro-simbólica do ARANDU proporciona ganhos substanciais de desempenho, mantendo alta precisão mesmo quando utilizado com modelos de linguagem mais compactos.

Tabela 7.1: Desempenho comparativo no dataset **WebQSP** utilizando o modelo **Llama 3.2 3B**.

Método	**Recall@10 (%)**	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	27.34	43.76	32.42	29.66
GraphRAG	36.84	55.07	35.87	26.96
G-Retriever	50.76	66.81	29.08	36.97
ARANDU	**54.91**	**70.62**	**36.59**	**43.91**

A Tabela 7.2 apresenta os resultados comparativos no conjunto WebQSP utilizando o modelo Llama 3.1 8B. Observa-se que o ARANDU supera os métodos de referência na maioria das métricas avaliadas, incluindo Recall@10, Hits@10 e F1-Score. Em particular, o ARANDU atinge um Recall@10 de 73,91% e um F1-Score de 53,55%, indicando maior capacidade de recuperar respostas corretas e gerar respostas mais precisas. Embora o G-Retriever tenha apresentado uma Precision@10 ligeiramente superior (58,34% versus 57,06%), o ARANDU demonstra desempenho superior nas demais métricas em relação ao G-Retriever (Recall@10 de 62,45% e F1-Score de 47,23%) e ao GraphRAG (Recall@10 de 40,82% e F1-Score de 22,16%). Esse resultado pode estar atribuído ao fato de que a etapa de orquestração lógica do ARANDU expande o subgrafo com inferências e regras lógicas. Esta expansão ocasionalmente introduz caminhos adicionais que, embora semanticamente relevantes, podem não corresponder exatamente às respostas de referência. Consequentemente, observa-se uma pequena redução na precisão das primeiras posições. Essa pequena redução na precisão é compensada pelo expressivo aumento na revocação, que é a métrica que mede a capacidade do sistema de recuperar todas as respostas corretas.

Esses resultados evidenciam o impacto positivo da integração dos módulos de recuperação híbrida, otimização estrutural e orquestração lógica. Os dados reforçam a hipótese de que a arquitetura neuro-simbólica proposta proporciona ganhos substanciais de desempenho. Este desempenho superior é mantido mesmo utilizando um modelo de linguagem de porte intermediário.

Tabela 7.2: Desempenho comparativo no dataset **WebQSP** utilizando o modelo **Llama 3.1 8B**.

Método	**Recall@10 (%)**	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	33.16	49.56	32.29	24.27
GraphRAG	40.82	60.54	24.76	22.16
G-Retriever	62.45	77.87	**58.34**	47.23
ARANDU	**73.91**	**79.50**	57.06	**53.55**

7.1.2 Resultados no Dataset MetaQA

A Tabela 7.3 apresenta os resultados no dataset MetaQA utilizando o modelo Llama 3.2 3B. Observa-se que, para questões 1-hop, tanto o G-Retriever quanto o ARANDU apresentam desempenhos elevados, com *Recall@10* acima de 90% e *Precision@10* acima de 94%. O G-Retriever atinge o valor máximo em *Hits@10* (100%) e F1-Score (96,87%), enquanto o ARANDU apresenta valores muito próximos (96,16% e 92,17%, respectivamente).

No entanto, ao analisar o cenário 2-hop, que é mais desafiador, o ARANDU supera os concorrentes em todas as métricas: *Recall@10* (74,58%), *Hits@10* (88,23%), *Precision@10* (71,56%) e *F1-Score* (70,58%). O G-Retriever, embora apresente bom desempenho, obteve resultados inferiores ao ARANDU em todos os indicadores. Já o GraphRAG apresenta resultados significativamente inferiores, especialmente em *Recall@10* (34,49%) e F1-Score (39,27%). Já para o cenário 3-hop, dada a complexidade maior, os resultados são inferiores para todos os métodos. Todavia, o ARANDU mantém desempenho superior aos outros métodos.

Ao análisar esses resultados, duas conclusões podem ser extraídas. Primeiro, a representação neural via GNN, presentes tanto no ARANDU quanto no G-Retriever, consegue melhorar significativamente a qualidade das respostas. Sua característica treinável permite que o conjunto como um todo obtenha melhores resultados. Segundo, o módulo de orquestração lógica, que o ARANDU possui a mais que o G-Retriever, melhora o desempenho em cenários de raciocínio multi-hop. Isso evidencia que a arquitetura proposta é mais robusta para questões que exigem raciocínio multi-hop, mantendo alta precisão e cobertura.

A Tabela 7.4 mostra os resultados para o mesmo dataset, agora utilizando o modelo Llama 3.1 8B, que possui maior capacidade. Nota-se que, para questões 1-hop, tanto G-Retriever quanto ARANDU mantêm desempenhos elevados e muito próximos, com *Recall@10* acima de 91%, *Hits@10* acima de 95% e *Precision@10* acima de 93%. Novamente, observa-se que os resultados do G-Retriever apresentam valores ligeiramente superiores em questões de 1-hop. Na comparação dos resultados das duas tabelas, curiosamente, observa-se que o modelo menor, de 3 bilhões de parâmetros,

Tabela 7.3: Desempenho comparativo no dataset **MetaQA** utilizando o modelo **Llama 3.2 3B**.

Método	**MetaQA - 1-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	48.90	61.20	66.03	52.93
GraphRAG	48.51	60.20	65.22	52.44
G-Retriever	**95.83**	**100.0**	**95.83**	**96.87**
ARANDU	91.89	96.16	94.71	92.17
Método	**MetaQA - 2-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	31.60	48.05	55.07	35.72
GraphRAG	34.49	53.56	61.37	39.27
G-Retriever	72.03	86.12	64.74	64.73
ARANDU	**74.58**	**88.23**	**71.56**	**70.58**
Método	**MetaQA - 3-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	22.96	31.83	30.04	26.08
GraphRAG	28.69	36.14	29.82	29.24
G-Retriever	43.22	51.67	38.84	40.91
ARANDU	**44.75**	**52.94**	**42.94**	**43.83**

conseguiu resultados melhores que o modelo maior, de 8 bilhões de parâmetros, em questões de 1-hop, com o G-Retriever e o ARANDU. Isso é observado em todas as métricas. Entretanto, o mesmo não ocorre para o desempenho dos dois modelos com o GraphRAG. Neste caso, os resultados do modelo menor foram inferiores ao modelo maior.

A divergência no desempenho pode ser atribuída a dois fatores principais. Primeiro, a forma como cada sistema apresenta o contexto ao modelo de linguagem é fundamental: o G-Retriever e o ARANDU combinam a representação linear (textual) com uma representação neural. Enriquecendo o *prompt* de entrada, simplificando a tarefa e favorecendo a precisão de um modelo menor (3B). Em contraste, o GraphRAG fornece o contexto como texto bruto, demandando uma maior capacidade de síntese que beneficia o modelo maior (8B). Segundo, o módulo de orquestração lógica do ARANDU, embora não prejudique o desempenho em questões simples de 1-hop (mantendo resultados competitivos de 91-95%), não oferece vantagem significativa nestes cenários, uma vez que o processamento adicional não se justifica quando a resposta pode ser obtida através de um único salto no grafo.

No cenário 2-hop, o ARANDU novamente se destaca, atingindo os maiores valores em todas as métricas: *Recall@10* (78,71%), *Hits@10* (90,67%), *Precision@10* (81,52%) e *F1-Score* (77,96%). O G-Retriever apresenta desempenho competitivo, mas

inferior ao ARANDU, enquanto o GraphRAG permanece com resultados significativamente mais baixos, especialmente em *Recall@10* (37,17%) e F1-Score (39,26%). Tal comportamento também é observado para o cenário 3-hop, todavia, com resultados inferiores para todos os métodos.

Tabela 7.4: Desempenho comparativo no dataset **MetaQA** utilizando o modelo **Llama 3.1 8B**.

Método	**MetaQA - 1-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	61.35	71.11	81.99	65.57
GraphRAG	64.13	73.61	83.53	68.12
G-Retriever	**92.89**	**96.13**	**95.01**	**92.92**
ARANDU	91.83	95.33	93.97	91.93
Método	**MetaQA - 2-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	31.53	44.33	47.95	33.63
GraphRAG	37.17	52.08	56.64	39.26
G-Retriever	75.01	88.91	77.09	73.13
ARANDU	**78.71**	**90.67**	**81.52**	**77.96**
Método	**MetaQA - 3-hop**			
	Recall@10 (%)	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
NaiveRAG	28.92	33.60	28.77	28.84
GraphRAG	34.69	42.14	33.82	34.24
G-Retriever	47.01	53.35	46.25	46.63
ARANDU	**52.23**	**54.40**	**50.91**	**51.56**

7.1.3 Discussão Sobre os Resultados

A análise consolidada dos resultados apresentados nas Tabelas 7.1 a 7.4 revela um padrão claro e consistente que responde diretamente à Questão de Pesquisa 1 (QP1). O *framework* ARANDU demonstra um desempenho superior em relação aos baselines na vasta maioria das métricas e cenários avaliados, posicionando-se como uma solução robusta e eficaz para a tarefa de QA em grafos de conhecimento.

A principal vantagem competitiva do ARANDU manifesta-se em cenários de raciocínio multi-hop, como evidenciado nos resultados do MetaQA 2-hop. Nestas tarefas complexas os resultados do ARANDU mostram superioridade, o que pode ser diretamente atribuído à sua arquitetura neuro-simbólica. A integração do módulo de orquestração lógica, em particular, permite uma otimização do subgrafo que se mostra importante para a inferência em múltiplos saltos, garantindo que o contexto fornecido ao LLM seja não apenas relevante, mas também logicamente coeso.

É igualmente instrutivo notar o desempenho em questões de 1-hop, onde o G-Retriever apresenta uma ligeira vantagem. Este resultado não diminui a eficácia do ARANDU, mas sim elucida uma característica importante do seu projeto: o mecanismo de raciocínio lógico, embora altamente benéfico para a complexidade, representa uma etapa computacional que não se traduz em ganhos para a extração de fatos diretos. O ARANDU se mantém altamente competitivo nestes cenários, mas sua arquitetura é otimizada para desafios de maior complexidade.

Também foi possível observar o impacto da qualidade do contexto na dependência do tamanho do modelo de linguagem. O fato de o ARANDU obter ganhos de desempenho expressivos com o modelo de 3B, e de a transição para o modelo de 8B resultar em melhorias apenas marginais, reforça a hipótese central desta tese. A arquitetura do ARANDU, ao fornecer um contexto de alta qualidade, reduz a carga inferencial sobre o LLM, permitindo que modelos menores atinjam um desempenho competitivo.

Em suma, os resultados validam empiricamente a arquitetura neuro-simbólica do ARANDU. A combinação sinérgica de recuperação híbrida, otimização estrutural, orquestração lógica e representação neural confere ao *framework* um equilíbrio superior entre precisão e revocação, respondendo afirmativamente à QP1.

7.2 Análise do Componente de Recuperação

Esta seção investiga a eficácia dos componentes de recuperação do framework ARANDU. O foco está na avaliação isolada da etapa de recuperação, anterior ao processo de geração de respostas. Esta abordagem permite uma compreensão detalhada da contribuição específica dos módulos de recuperação textual e estruturada e do módulo de orquestração lógica da arquitetura proposta.

Como visto no Capítulo 4, o framework ARANDU implementa uma abordagem híbrida de recuperação. Esta estratégia integra três módulos complementares para recuperação de evidências: o módulo de recuperação textual, o módulo de recuperação estruturada e o módulo de orquestração lógica. A arquitetura modular visa superar limitações das abordagens convencionais que utizam apenas recuperação textual ou estrutural isoladamente.

Como abordado na seção 6.5, a metodologia de avaliação da recuperação emprega métricas específicas. Estas quantificam tanto a capacidade de identificação quanto a qualidade do ranqueamento das evidências relevantes. As métricas Hits@K e Recall@K avaliam a presença de informações corretas nos top-K itens recuperados. Por sua vez, nDCG@K e MRR (Mean Reciprocal Rank) medem a qualidade do posicionamento dessas informações no ranqueamento final. Esta abordagem permite uma análise granular da

eficácia dos componentes de recuperação. A análise é independente do desempenho do módulo de geração.

Os experimentos foram conduzidos nos datasets WebQSP e MetaQA. Esta configuração permite validação em diferentes domínios e complexidades de perguntas. A comparação com baselines estabelecidos fornece contexto para avaliar os ganhos obtidos pela estratégia híbrida proposta. Os baselines incluem abordagens de RAG tradicional, GraphRAG e o G-Retriever. As análises subsequentes demonstram como a integração sinérgica entre recuperação densa e otimização estrutural resulta em melhorias consistentes na qualidade das evidências recuperadas.

7.2.1 Resultados no Dataset WebQSP

A avaliação da etapa de recuperação é fundamental para compreender a eficácia dos componentes de recuperação híbrida do ARANDU antes da geração da resposta final. Esta análise foca especificamente na capacidade do sistema de identificar e recuperar informações relevantes do grafo de conhecimento, independentemente da qualidade da geração posterior. As métricas utilizadas incluem Recall@K, que mede a proporção de respostas corretas presentes nos top-K itens recuperados, e Mean Reciprocal Rank (MRR), que avalia a qualidade do ranqueamento dos itens relevantes.

Os resultados da etapa de recuperação demonstram que a combinação dos módulos de recuperação textual, estruturada e de orquestração lógica proporciona ganhos significativos em relação às abordagens tradicionais. Esta sinergia resulta em conjuntos de evidências mais informativos e contextualmente ricos, estabelecendo uma base sólida para as etapas subsequentes do *pipeline*.

A Tabela 7.5 apresenta os resultados específicos da etapa de recuperação no conjunto WebQSP, isolando o desempenho dos componentes de recuperação antes da geração da resposta final. Como é possível observar na tabela, os resultados do G-Retriever e GraphRAG estão expostos em uma única linha. Isso se dá pelo fato de que, para os experimentos propostos, o G-Retriever e GraphRAG utilizam a mesma abordagem de recuperação. Os dados demonstram que o ARANDU supera o G-Retriever em todas as métricas de recuperação avaliadas. Especificamente, o ARANDU alcança um Hits@10 de 57,83%, comparado aos 50,45% do G-Retriever, representando um ganho de aproximadamente 14,6%. O Recall@10 do ARANDU atinge 44,45%, superando os 37,51% do G-Retriever por uma margem de 18,5%. A métrica nDCG@10, que avalia a qualidade do ranqueamento dos itens recuperados, mostra um desempenho particularmente forte do ARANDU com 70,22%, substancialmente superior aos 47,68% do G-Retriever. O Mean Reciprocal Rank (MRR) do ARANDU também é superior, atingindo 36,56% comparado aos 25,73% do G-Retriever.

Uma análise complementar dos resultados revela que, embora o ARANDU supere o G-Retriever em Hits@20 (64,52% versus 62,49%) e Recall@20 (52,73% versus 50,48%), os ganhos percentuais são menos expressivos quando comparados aos observados em Hits@10 e Recall@10. Esta observação indica que o *pipeline* proposto pelo ARANDU demonstra particular eficácia em posicionar evidências relevantes nas primeiras posições do ranqueamento. A capacidade de priorizar informações de alta qualidade nos primeiros resultados é especialmente valiosa em aplicações práticas, onde a eficiência computacional e a rapidez de resposta são fatores determinantes para a experiência do usuário.

Tabela 7.5: Desempenho da etapa de recuperação no dataset **WebQSP**.

Método	Hits@10 (%)	Hits@20 (%)	Recall@10 (%)	Recall@20 (%)	Precision@10 (%)	nDCG@10 (%)	MRR (%)
NaiveRAG	38.83	46.98	27.89	35.90	17.03	49.36	26.72
G-Retriever/GraphRAG	50.45	62.49	37.51	50.48	18.38	47.68	25.73
ARANDU	**57.83**	**64.52**	**44.45**	**52.73**	**23.94**	**70.22**	**36.56**

Estes resultados validam a eficácia da abordagem híbrida de recuperação, combinada à orquestração lógica proposta. A superioridade do ARANDU nas métricas de recuperação estabelece uma base sólida para o desempenho superior observado na etapa de geração, confirmando que a qualidade da recuperação é um fator determinante para o sucesso geral do sistema.

7.2.2 Resultados no Dataset MetaQA

A Tabela 7.6 apresenta os resultados da etapa de recuperação no dataset **MetaQA**, tanto para questões do tipo 1-hop quanto 2-hop. No cenário 1-hop, todos os métodos apresentam desempenhos elevados em *Hits@10* e *Recall@10*. O ARANDU atinge 99,05% e 96,80%, respectivamente, superando o G-Retriever (98,49% e 96,26%). Embora o NaiveRAG apresente o melhor desempenho em *Recall@10* (98,38%) e MRR (84,87%), o ARANDU se destaca de forma expressiva na métrica nDCG@10 (95,21%), superando significativamente o G-Retriever (64,16%) e o NaiveRAG (87,46%). Em termos de precisão, os três métodos apresentam desempenhos similares, com o G-Retriever ligeiramente superior (47,11%) em relação ao ARANDU (46,75%) e NaiveRAG (46,54%). Isso indica que, embora o NaiveRAG seja eficaz na recuperação de respostas corretas em cenários simples, o ARANDU oferece um ranqueamento mais refinado das evidências. Esta capacidade reflete uma ordenação mais útil para o usuário final.

No cenário 2-hop, que envolve questões mais complexas, a diferença entre os métodos se acentua. O ARANDU mantém a liderança em todas as métricas avaliadas. Em *Hits@10*, o ARANDU alcança 85,72% contra 84,84% do G-Retriever. No *Recall@10*, obtém 71,27% versus 69,76%. A *Precision@10* do ARANDU atinge 28,95%, superando os 25,29% do G-Retriever. O ganho em nDCG@10 é substancial: 61,71% contra 43,77%

do baseline. O MRR apresenta diferença ainda mais expressiva: 49,66% versus 24,00%. O ganho em nDCG@10 e MRR é especialmente relevante. Isso evidencia que o ARANDU é mais eficaz em priorizar respostas corretas nas primeiras posições.

Esses resultados consolidam a percepção de que a abordagem neuro-simbólica do ARANDU, que combina a otimização estrutural e a orquestração lógica, proporciona ganhos substanciais na etapa de recuperação. Tais ganhos são observados especialmente em cenários de maior complexidade. O desempenho superior em métricas de ranqueamento reforça a robustez do método para aplicações práticas.

Tabela 7.6: Desempenho da etapa de recuperação no dataset **MetaQA**.

Método	**MetaQA - 1-hop**				
	Hits@10 (%)	**Recall@10 (%)**	**Precision@10 (%)**	**nDCG@10 (%)**	**MRR (%)**
NaiveRAG	99.02	**98.38**	46.54	87.46	**84.87**
G-Retriever/GraphRAG	98.49	96.26	**47.11**	64.16	54.15
ARANDU	**99.05**	96.80	46.75	**95.21**	79.33
Método	**MetaQA - 2-hop**				
	Hits@10 (%)	**Recall@10 (%)**	**Precision@10 (%)**	**nDCG@10 (%)**	**MRR (%)**
NaiveRAG	74.39	60.29	18.38	41.36	24.92
G-Retriever/GraphRAG	84.84	69.76	25.29	43.77	24.00
ARANDU	**85.72**	**71.27**	**28.95**	**61.71**	**49.66**
Método	**MetaQA - 3-hop**				
	Hits@10 (%)	**Recall@10 (%)**	**Precision@10 (%)**	**nDCG@10 (%)**	**MRR (%)**
NaiveRAG	37.51	30.93	17.00	22.97	15.72
G-Retriever/GraphRAG	53.58	44.18	24.29	32.81	22.46
ARANDU	**58.86**	**49.37**	**27.72**	**36.04**	**36.34**

7.2.3 Análise Qualitativa e Implicações da Etapa de Recuperação

Os resultados quantitativos da etapa de recuperação revelam um padrão consistente: embora os ganhos em métricas de cobertura (Hits@K, Recall@K) sejam significativos, a superioridade do ARANDU é mais pronunciada em métricas de qualidade de ranqueamento (nDCG@K, MRR).

É importante notar que a técnica adotada para recuperação estruturada é semelhante para o GraphRAG, G-Retriever e ARANDU. Todos utilizam o algoritmo PCST para a otimização estruturada do subgrafo. Portanto, a vantagem competitiva do ARANDU não reside na adoção do PCST em si, mas na fase de orquestração lógica subsequente. O desempenho superior do ARANDU demonstra o impacto positivo dos módulos de enriquecimento lógico e ponderação de caminhos.

A fase de orquestração lógica explica diretamente os ganhos expressivos em nDCG e MRR. Ao reduzir o ruído lógico, o ARANDU produz um subgrafo final que representa uma "explicação"mais coerente e fiel à pergunta. O ranqueamento das

evidências reflete essa coerência aprimorada, posicionando os fatos mais relevantes nas primeiras posições. A ampliação da vantagem em cenários 2-hop, onde o ganho em MRR supera 100% (de 24,00% para 49,66%), valida esta interpretação. À medida que a complexidade da consulta aumenta, a probabilidade de extrair caminhos espúrios cresce, tornando a validação lógica progressivamente mais importante.

As implicações para o sistema como um todo são diretas, conforme discutido na Seção 7.1. Fornecer ao LLM um contexto que foi otimizado estruturalmente (via PCST), enriquecido e ponderado logicamente reduz a necessidade de uma carga elevada. O modelo de linguagem recebe um conjunto de evidências de alta qualidade, permitindo que se concentre na tarefa de síntese e geração. É possível observar, então, que a análise da etapa de recuperação, apresentada nesta seção, responde a questão de pesquisa 2 (QP2).

7.3 Análise da Operacionalização Estrutural via GNN

Esta seção aborda a Questão de Pesquisa 3 (QP3): *"Como a representação otimizada do subgrafo, operacionalizada através de Graph Neural Networks, afeta a geração da resposta pelo LLM?"*. O módulo de representação neural da arquitetura ARANDU postula que a conversão do subgrafo de evidências em uma representação neural via GNN, preservando a topologia, é mais eficaz do que a simples linearização textual. Para validar esta hipótese, foi realizado um estudo de ablação focado na etapa de geração de respostas.

A metodologia compara o desempenho de duas configurações do *framework*. A primeira, **ARANDU (Completo)**, utiliza o pipeline integral, onde a GNN (módulo de representação neural) codifica o subgrafo refinado em *embeddings* ricos em estrutura, que são então utilizados para guiar o LLM. A segunda, **ARANDU (sem GNN)**, remove este componente. Nesta versão, o mesmo subgrafo refinado pelo módulo de representação neural é serializado em formato de triplos textuais (e.g., `(entidade_cabeça, relação, entidade_cauda)`). Este texto é então concatenado ao *prompt* do LLM, uma abordagem comum em sistemas GraphRAG. A comparação foi realizada no dataset WebQSP, utilizando o modelo Llama 3.1 8B, avaliando as métricas de geração de resposta final.

A Tabela 7.7 apresenta os resultados do estudo. Os dados indicam que a operacionalização da estrutura via GNN proporciona melhorias consistentes em todas as métricas avaliadas. O ganho é expressivo, principalmente na métrica de precisão (de 29.92 para 57.06).

A resposta à QP3 pode ser articulada a partir desses resultados. A operacionalização estrutural via GNN impacta positivamente a qualidade da resposta final ao fornecer ao LLM uma representação mais rica e fiel do contexto recuperado. A linearização textual, embora informativa, resulta em perda de informação topológica. Relações comple-

Tabela 7.7: Resultados do estudo de ablação do componente GNN na etapa de geração, utilizando o dataset **WebQSP** e o modelo **Llama 3.1 8B**.

Método	**Recall@10 (%)**	**Hits@10 (%)**	**Precision@10 (%)**	**F1-Score (%)**
ARANDU (sem GNN)	50.55	65.51	42.92	46.42
ARANDU (Completo)	**73.91**	**79.50**	**57.06**	**53.55**

xas, como múltiplos nós compartilhando uma mesma conexão, são difíceis de capturar em uma sequência de texto. A GNN, por outro lado, processa o grafo em seu formato nativo, e seus *embeddings* de saída encapsulam tanto a semântica dos nós quanto a estrutura das suas interconexões.

Adicionalmente, durante os experimentos foi possível observar que, com o uso da representação neural através da GNN, as respostas geradas pelo LLM se tornaram mais objetivas. Isso garante que a resposta seja mais focada nas respostas esperadas, gerando menos texto desnecessário. A análise confirma que, embora a recuperação e o refinamento lógico sejam os fatores mais determinantes, a forma como o contexto estruturado é apresentado ao LLM também é um componente relevante para otimizar o desempenho. A abordagem neuro-simbólica, que combina a lógica explícita com a representação neural implícita, se mostra, portanto, mais robusta que alternativas baseadas apenas em texto.

7.4 Análise de Eficiência Computacional

Como visto na Seção 6.1, a quarta questão de pesquisa (QP4) busca investigar se o *framework* ARANDU mantém uma eficiência computacional compatível com seu princípio de design de ser uma solução "leve". A análise para essa questão de pesquisa envolve não apenas a comparação do *framework* com outras abordagens, como também o estudo do consumo de recursos e tempo de cada módulo.

As métricas de tempo utilizadas foram vazão (questões respondidas por segundo) e latência (tempo médio para responder uma única pergunta). Importante notar que nem sempre a vazão foi o inverso da latência, uma vez que algumas etapas foram executadas em lote.

Em relação ao consumo de recursos, as métricas utilizadas foram: porcentagem de uso da CPU e GPU e quantidade de memória RAM e VRAM utilizadas. É relevante avaliar tanto o ambiente da CPU quanto o da GPU, uma vez que algumas tarefas são executadas fora da GPU, como é o caso da busca lexical, por exemplo. Todos os experimentos de eficiência foram conduzidos em um ambiente controlado, conforme descrito na Seção 6.4. O dataset utilizado como referência foi o WebQSP.

7.4.1 Etapa de Preparação

A etapa de preparação, embora executada offline e apenas uma vez, é um componente cujo custo computacional precisa ser analisado, especialmente em cenários com grandes volumes de dados. Esta fase é responsável pela criação dos artefatos de conhecimento que sustentam todo o pipeline de execução, incluindo a construção do grafo de conhecimento, a geração de índices para busca vetorial e lexical, e a mineração de regras lógicas. A Tabela 7.8 apresenta o tempo de execução e o consumo de recursos para cada uma dessas sub-etapas.

Tabela 7.8: Métricas de performance para a etapa de preparação (Dataset WebQSP)

Sub-etapa	Tempo (min)	Pico CPU (%)	Pico RAM (GB)	Pico GPU (%)	Pico VRAM (GB)
Construção do Grafo	16	55	2,6	-	-
Indexação Vetorial	22	8	-	85	1,4
Indexação Lexical	14	70	1,8	-	-
Mineração de Regras	25	92	3,4	-	-
Total	73	92	3,4	85	1,4

A análise dos dados de performance revela que a mineração de regras lógicas representa o maior custo computacional da etapa de preparação, consumindo 25 minutos (34% do tempo total) e atingindo o pico de utilização de CPU (92%) e RAM (3,4 GB). A indexação vetorial, embora utilize recursos de GPU, apresenta o menor impacto no tempo total devido à paralelização eficiente. O consumo de memória VRAM ficou sempre abaixo de 1,5 GB, permitindo que tal processo possa ser realizado inclusive em GPUs de entrada. A construção do grafo de conhecimento demonstra boa eficiência, processando grandes volumes de dados em apenas 16 minutos com consumo moderado de recursos. O tempo total de 77 minutos para preparação completa é aceitável considerando que esta etapa é executada apenas uma vez, estabelecendo a base de conhecimento para todas as consultas subsequentes.

É importante destacar que alguns processos podem ser realizados em paralelo, como a construção do grafo de conhecimento e a indexação vetorial. A análise apresentada na Tabela 7.8 considera apenas a execução sequencial de cada sub-etapa. O experimento foi executado dessa forma, uma vez que o objetivo era avaliar o custo computacional de cada sub-etapa isoladamente. Durante a construção do ambiente, foi possível observar que toda essa etapa offline pode levar menos de uma hora (60 minutos), caso alguns processos sejam realizados em paralelo.

7.4.2 Etapa de Treinamento

Um dos aspectos de design do ARANDU é a eficiência computacional, que se manifesta de forma proeminente na etapa de treinamento. Conforme detalhado na

Seção 5.4.5, a estratégia adotada consiste em manter os pesos do LLM congelados, treinando exclusivamente os parâmetros da GNN e de sua camada de projeção. Essa abordagem anula o custo associado ao ajuste fino de um LLM, que demandaria recursos computacionais significativamente maiores, e foca o aprendizado na geração de *soft prompts* vetoriais que o LLM pré-treinado possa interpretar. A Tabela 7.9 quantifica o custo computacional desta etapa para as diferentes configurações de LLMs utilizadas nos experimentos. Como o G-Retriever utiliza uma abordagem semelhante, ele foi incluído na tabela para referência e comparação.

Tabela 7.9: Métricas de performance para a etapa de treinamento (Dataset WebQSP)

Configuração	Tempo/Época (s)	Tempo/Lote (s)	Tamanho do Lote	Pico VRAM (GB)	Uso GPU (%)	Pico RAM (GB)	Uso CPU (%)
ARANDU (Llama 3.2 3B)	410	0,87	6	10,8	94	1,9	5,4
ARANDU (Llama 3.1 8B)	1308	1,36	3	10,2	97	2,1	5,2
G-Retriever (Llama 3.1 8B)	2111	1,50	2	11,0	97	2,1	5,2

Como é possível observar na Tabela 7.9, o ARANDU apresentou tempo de treinamento significativamente menor que o obtido pelo G-Retriever. Graças a mecanismos de reuso de *tokens* e à utilização de *chat template*, o ARANDU consegue ser mais eficiente no uso de memória VRAM. Assim, liberando espaço para que seja possível trabalhar com lotes maiores (com Llama 3.1 8B, o lote no ARANDU foi de 3 perguntas, enquanto no G-Retriever foi de 2 perguntas), aumentando a vazão e a eficiência do sistema durante a etapa de treinamento.

7.4.3 Etapa de Execução

A etapa de execução representa o pipeline online do ARANDU, sendo a fase mais crítica do ponto de vista da eficiência percebida pelo usuário final. Nesta seção, é apresentada uma análise detalhada da latência e do consumo de recursos de cada um dos módulos que compõem este pipeline, desde a recuperação inicial de evidências até a geração da resposta final pelo LLM. A Tabela 7.10 sumariza os valores médios observados para cada componente, permitindo identificar os principais gargalos computacionais e avaliar o desempenho ponta a ponta do framework em um cenário de inferência em tempo real.

A análise dos resultados apresentados na Tabela 7.10 revela características importantes do pipeline de execução do ARANDU. O módulo de LLM representa o principal gargalo computacional, consumindo 740ms dos 1036ms totais de latência (71,4% do tempo total). Este comportamento é esperado, considerando a natureza intensiva da geração de texto por modelos de linguagem de grande escala. A recuperação estruturada via PCST constitui o segundo maior consumidor de tempo (164ms), refletindo a complexidade algorítmica da busca em grafos. A velocidade desse módulo está diretamente relacionada ao tamanho do grafo de conhecimento, como será analisado na Seção 7.4.4.

Tabela 7.10: Valores médios em métricas de performance por módulo do pipeline (WebQSP e Llama 3.1 8B)

Módulo	Latência (ms)	Pico GPU (%)	Pico VRAM (MB)	Pico CPU (%)	Pico RAM (MB)
Recuperação Textual	47,5	8	310	8	10
- Lexical (BM25)	28	-	-	8	10
- Semântica (embeddings)	19	8	310	-	10
- Fusão (RRF)	0,5	-	-	5,5	-
Recuperação Estruturada (PCST)	164	-	-	60	40
Enriquecimento Lógico	20	-	-	9,5	8,5
Ponderação de Caminhos	18	-	-	28	6,7
GNN (GATv2)	12	-	-	80	5,5
LLM (Llama 3.1 8B)	740	95	6650	40	-
Fim-a-fim	1036	95	6650	80	50

Os demais módulos apresentam latências reduzidas, com destaque para a eficiência da fusão RRF (0,5ms) e do processamento por GNN (12ms).

Em termos de recursos computacionais, observa-se distribuição equilibrada entre processamento em GPU e CPU. O LLM concentra o uso intensivo de GPU (95%) e VRAM (6650MB), enquanto módulos como PCST e GNN utilizam predominantemente CPU. Neste ponto, é importante destacar que a GNN também pode ser executada em GPU. Entretanto, em testes realizados, constatou-se que, para a configuração de hardware apresentada, não há perdas de performance significativas ao executar a GNN em CPU. Em contrapartida, há uma redução média do uso de VRAM em torno de 100MB. Com isso, optou-se por executar a GNN em CPU. Esta arquitetura híbrida permite otimização de recursos e redução de contenção por memória GPU. O consumo total de RAM (50MB) permanece moderado, indicando eficiência na gestão de memória do sistema.

Para contextualizar a eficiência do ARANDU, é fundamental comparar seu desempenho com o de outras abordagens de RAG sobre grafos de conhecimento. Esta análise posiciona o framework em relação a baselines representativos, permitindo uma avaliação objetiva de suas vantagens e compromissos em termos de custo computacional. A Tabela 7.11 apresenta uma comparação de métricas de performance fim-a-fim entre o ARANDU (em suas duas configurações de LLM) e os baselines selecionados.

Tabela 7.11: Comparação de performance fim-a-fim com baselines (Dataset WebQSP e Llama 3.1 8B)

Abordagem	Latência (ms)	Vazão (q/s)	Média Uso GPU (%)	Pico VRAM (MB)	Média Uso CPU (%)	Pico RAM (MB)
NaiveRAG	641	1,56	91	6947	5	1920
GraphRAG	953	1,18	92	6850	8	2000
G-Retriever	7212	0,28	95	7235	8	2085
ARANDU	1036	1,12	92	6650	9	2140

Os resultados apresentados na Tabela 7.11 revelam aspectos importantes sobre o posicionamento do ARANDU em relação aos baselines estabelecidos. O framework demonstra latência competitiva, com um tempo médio de 1036 milissegundos (ms),

posicionando-se próximo do GraphRAG (953 ms) e bem abaixo do G-Retriever (7712 ms). Nota-se que o G-Retriever apresenta dados de latência e vazão significativamente piores que o ARANDU e os outros baselines. A principal diferença está na forma de utilização do modelo de linguagem.

A abordagem do G-Retriever é baseada em concatenação de tokens textuais, enquanto o ARANDU utiliza *chat template*. Esta diferença metodológica impacta significativamente a eficiência computacional. O *chat template* estrutura a entrada de forma otimizada para o modelo, permitindo processamento mais eficiente. A concatenação textual, por sua vez, resulta em sequências menos estruturadas que demandam maior processamento. Além disso, o *chat template* facilita a tokenização e reduz a ambiguidade na interpretação do contexto.

Curiosamente, o tempo de inferência do G-Retriever é, em média, maior que o de treinamento. Este comportamento decorre das diferenças fundamentais entre os processos de treinamento e inferência. Durante o treinamento, o modelo processa sequências completas em paralelo através de operações matriciais. Na inferência, a geração de texto requer decodificação sequencial autorregressiva, onde cada token é gerado condicionalmente aos anteriores, aumentando a latência e reduzindo a vazão. Essa característica é ainda mais evidente no G-Retriever, por utilizar concatenação textual e, com isso, precisar processar uma maior quantidade de tokens para cada questão.

Embora apresente maior tempo de resposta que abordagens mais simples, o ARANDU mantém vazão adequada de 1,12 questão por segundo (q/s). O uso médio de GPU permanece consistente com os baselines (92%), indicando eficiência similar no processamento paralelo. Destaca-se o menor pico de VRAM (6650MB) entre todas as abordagens e o maior pico de RAM (2140MB), com uma diferença pequena em relação ao G-Retriever e GraphRAG. Essa característica é particularmente relevante, uma vez que alguns dos módulos do ARANDU são executados em CPU (como a orquestração lógica e a recuperação estruturada). O aumento do uso de CPU e RAM não é significativo, em comparação à redução do uso de GPU (relacionado com respostas mais objetivas e com menor quantidade de tokens). Estes resultados indicam que o ARANDU oferece compromisso equilibrado entre qualidade de resposta e eficiência computacional.

7.4.4 Análise de Escalabilidade

A capacidade de um sistema manter seu desempenho à medida que o volume de dados aumenta é um indicador fundamental de sua robustez e aplicabilidade em cenários reais. Para avaliar a escalabilidade do ARANDU, foi investigado como a latência de inferência do pipeline de execução online responde ao crescimento do Grafo de Conhecimento.

Para esta análise, o dataset MetaQA foi selecionado como referência, pois sua estrutura dividida em subconjuntos de complexidade crescente (1 salto (*1-hop*), 2 saltos (*2-hop*) e 3 saltos (*3-hop*)) permite simular grafos de diferentes tamanhos de forma controlada. Ao executar o pipeline completo sobre cada um desses subconjuntos, é possível medir o impacto do aumento do número de nós e triplas no tempo de resposta. A Tabela 7.12 apresenta alguns dados estatísticos para cada subconjunto, como média, percentil 90%, desvio padrão e mediana. A partir desses dados, foram selecionados 10 exemplos com quantidade de nós e triplas próximos à média, à mediana e ao percentil de 90%

Tabela 7.12: Resultados de Escalabilidade do ARANDU no MetaQA.

Complexidade	**Triplas**	**Nós**
1-hop	Média: 7,7 | DP: 5,8	Média: 5,7 | DP: 2,9
	P90: 14 | Mediana: 7	P90: 8 | Mediana: 6
2-hop	Média: 682,4 | DP: 1658,2	Média: 615,6 | DP: 1530,6
	P90: 4092 | Mediana: 21	P90: 4033 | Mediana: 8
3-hop	Média: 20674,3 | DP: 18905,2	Média: 2744.8 | DP: 2424
	P90: 45247 | Mediana: 12523	P90: 5732 | Mediana: 1801

Tabela 7.13: Análise de escalabilidade: latência em função do tamanho do grafo

Complexidade	**Qtd Triplas**	**Qtd Nós**	**Fase I** **Recuperação (ms)**	**Fase II** **Orquestração Lógica (ms)**	**Fase III** **Geração (ms)**	**Tempo** **Total (ms)**
1-hop	8	6	2,7	64	749	816
1-hop	14	8	2,8	72	758	833
2-hop	21	8	3,2	75	795	873
2-hop	652 a 750	586 a 650	18,7	106	742	867
2-hop	4052 a 4182	4015 a 4045	72	196	782	1050
3-hop	20508 a 20700	2144 a 9119	182	289	755	1226
3-hop	45072 a 45374	4977 a 5250	254	396	742	1392

Os resultados da Tabela 7.13 demonstram comportamento escalável do ARANDU. A fase de recuperação apresenta crescimento linear com o tamanho do grafo, variando de 2,7 ms para grafos pequenos até 254 ms para grafos com mais de 45 mil triplas. A orquestração lógica mantém latência controlada, aumentando de 64 ms para 196 ms mesmo com crescimento significativo do grafo. Neste ponto, foi possível constatar que grafos com poucos tipos de arestas, como é o caso do MetaQA, tendem a ter uma latência maior na etapa de orquestração lógica. Isso ocorre porque as regras lógicas são mais específicas, sendo que para um mesmo grafo, quase todas as regras são aplicadas, o que aumenta a latência. Em grafos com mais tipos de arestas, como é o caso do WebQSP, a latência da etapa de orquestração lógica é menor. Isso se dá porque as regras são mais variadas, sendo que para um mesmo grafo, menos regras são aplicadas, o que diminui a latência.

A fase de geração permanece estável entre 742 ms e 795 ms, independentemente do tamanho do grafo. Estes resultados indicam que o framework mantém desempenho adequado mesmo com o aumento substancial da complexidade estrutural.

7.4.5 Discussão sobre os resultados em eficiência

Com base nas análises realizadas, esta subseção sintetiza os resultados de eficiência. A avaliação abordou latência, consumo de recursos e escalabilidade do framework ARANDU. Os resultados indicam que a fase online do sistema possui baixa latência e consumo computacional reduzido. Em contrapartida, a fase offline de indexação e treinamento demanda mais recursos. No entanto, este é um custo inicial que garante o bom desempenho em tempo de consulta. A escalabilidade se mostrou robusta, com tempo de resposta estável, mesmo com o crescimento do grafo.

Diante do exposto, conclui-se que o ARANDU é um framework leve e eficiente, respondendo afirmativamente à QP4. Essa eficiência se manifesta principalmente na sua operação online, que é o foco da interação com o usuário. A arquitetura foi projetada para que o custo computacional mais elevado ficasse restrito ao processamento offline. Tal decisão de projeto assegura uma experiência de uso ágil e com respostas rápidas.

A análise, contudo, revelou um importante compromisso entre latência e qualidade da resposta. O módulo de orquestração lógica pode introduzir uma latência adicional significativa em grafos com poucos tipos de arestas. No entanto, este componente é essencial para a geração de respostas mais completas e explicativas.

Adicionalmente, o compromisso entre o custo offline e a eficiência online é uma decisão de projeto fundamental. O investimento em uma indexação robusta e no treinamento de modelos GNN evita uso excessivo de recursos e tempo no momento de execução. Essa estratégia consolida o ARANDU como uma solução prática para sistemas de QA. O framework se mostra escalável para operar sobre grandes grafos de conhecimento, mantendo a agilidade.

7.5 Estudo de Caso da Execução do *Framework* ARANDU

Enquanto os resultados quantitativos apresentados nas seções anteriores demonstram a eficácia do framework ARANDU em métricas agregadas, esta seção se aprofunda em uma análise qualitativa e granular. O objetivo é fornecer uma demonstração concreta e passo a passo do pipeline em ação, rastreando o percurso de uma única consulta desde sua submissão até a geração da resposta final. Este exame detalhado serve para desmistificar

os mecanismos internos da arquitetura, ilustrando como cada componente contribui para o resultado.

Além de demonstrar a execução do pipeline, este estudo de caso tem como objetivo central evidenciar a explicabilidade e a rastreabilidade inerentes ao *design* neuro-simbólico do ARANDU. Ao tornar cada etapa do processamento da informação, da recuperação híbrida inicial à orquestração lógica do contexto, explícita e auditável, o *framework* permite uma clara verificação do caminho de raciocínio. Esta abordagem contrasta com sistemas puramente neurais, onde o processo inferencial permanece opaco. A análise a seguir demonstra como a sinergia entre os componentes neurais e simbólicos opera na prática. Esta combinação resulta não apenas em uma resposta correta, mas em uma resposta fundamentada. A fundamentação pode ser inspecionada e auditada em cada etapa do processo de raciocínio.

7.5.1 Seleção do Exemplo

A pergunta selecionada para a análise é a seguinte:

> **Consulta Exemplo:** *"What character did Natalie Portman play in Star Wars?"* (Que personagem Natalie Portman interpretou em Star Wars?)

A escolha desta consulta em particular não é fortuita. Ela foi motivada por três características que a tornam um excelente caso de teste para a hipótese central desta tese:

1. **Exigência de Raciocínio Multi-Salto:** A resposta não está diretamente conectada às entidades iniciais ("Natalie Portman", "Star Wars"). É necessário um raciocínio que atravesse o grafo: é esperado que o sistema encontre um caminho que conecte `Natalie Portman` aos filmes da franquia `Star Wars` que ela atuou. A partir dessa relação de atuação, identificar o personagem interpretado (e.g., `Padmé Amidala`). Este percurso `Ator` $\rightarrow$ `atuou em` $\rightarrow$ `Filme` $\rightarrow$ `retrata` $\rightarrow$ `Personagem` testa a capacidade do framework de construir e seguir uma cadeia de inferência precisa.
2. **Potencial de Falha em Abordagens Convencionais:** A consulta expõe a fragilidade de baselines menos robustos. Um `NaiveRAG`, focado em similaridade semântica, provavelmente recuperaria evidências sobre a filmografia geral de Natalie Portman e sobre o universo de Star Wars, mas poderia facilmente falhar em extrair a tripla específica que conecta a atriz ao seu personagem. Um `GraphRAG` sem otimização poderia recuperar um subgrafo ruidoso, incluindo outros atores da franquia ou outros filmes da atriz, poluindo o contexto e desviando o LLM da resposta correta.
3. **Clareza para Demonstração de Explicabilidade:** A cadeia de raciocínio necessária para responder a esta pergunta, embora multi-salto, é intuitiva e linear. Isso a torna ideal para demonstrar a rastreabilidade do pipeline do ARANDU. Podemos

visualizar com clareza como o subgrafo de evidências é construído, como a orquestração lógica prioriza o caminho correto em detrimento de outros, e como a resposta final é diretamente fundamentada nesta cadeia de evidências validada.

A partir dessa escolha, a análise subsequente detalhará, passo a passo, como o ARANDU processa esta consulta, desde a recuperação inicial de evidências até a geração da resposta fundamentada.

7.5.2 Demonstração do Pipeline: Passo a Passo

Nesta subseção, detalha-se a execução da consulta exemplo através do pipeline do ARANDU. O objetivo é ilustrar como as evidências são progressivamente coletadas, refinadas e estruturadas para fundamentar a resposta final.

Fase I: Recuperação Híbrida de Evidências

O objetivo desta fase inicial é identificar e extrair um subgrafo de evidências que seja conciso, relevante e estruturalmente coeso, servindo como a matéria-prima para o raciocínio subsequente.

1. Módulo de Recuperação Textual (TextRetriever)

- **Entrada:** A pergunta do usuário: *"What character did Natalie Portman play in Star Wars?"*.
- **Processo:** O `TextRetriever` utiliza sua abordagem híbrida para consultar os índices previamente construídos. A consulta é vetorizada para a busca semântica, enquanto os termos-chave ("Natalie Portman", "Star Wars") são utilizados para a busca lexical. O sistema identifica um conjunto inicial de entidades e relações com alta similaridade.
- **Saída (Candidatos Iniciais):** O módulo retorna uma lista ranqueada de entidades e relações candidatas. Esta saída serve como conjunto de "sementes"para a etapa subsequente. As Tabelas 7.14 e 7.15 apresentam as entidades e relações selecionadas.
- **Insight de Explicabilidade:** A rastreabilidade do processo inicia-se com a identificação de um conjunto de candidatos baseados em evidência semântica e lexical. É possível, neste ponto, compreender por que o sistema foca em determinados elementos do grafo. Contudo, o contexto ainda se apresenta como uma lista desestruturada e potencialmente ruidosa. No caso em questão, a entidade “the stars of star wars” pode ser considerada ruído. É importante observar que essa recuperação textual é realizada de forma híbrida. Ou seja, a recuperação semântica (*embeddings*)

é combinada com a recuperação léxica. Durante os experimentos foi possível observar que tal fusão garante resultados melhores, ao evidenciar entidades dentro do texto da pergunta.

Tabela 7.14: Entidades selecionadas pelo módulo de recuperação textual

ID do Nó	Texto do Nó
162	natalie portman
1094	star wars episode i: the phantom menace
1227	the stars of star wars

Tabela 7.15: Relações selecionadas pelo módulo de recuperação textual

Tipo da Relação
film_performance_character
film_film_character_portrayed_in_films
film_performance_actor
film_person_or_entity_appearing_in_film_films
film_film_starring

2. Módulo de Recuperação Estruturada (GraphRetriever)

- **Entrada:** A lista de candidatos iniciais e seus respectivos scores de relevância.
- **Processo:** O `GraphRetriever` formula um problema de *Prize-Collecting Steiner Tree* (PCST). As entidades e relações candidatas são designadas como "prêmios"a serem coletados, enquanto as arestas do grafo possuem "custos". O algoritmo busca o subgrafo conectado de menor custo que maximiza a coleta de prêmios.
- **Saída (Subgrafo Coeso):** O algoritmo PCST descarta nós de menor prêmio que não são essenciais para conectar os nós de prêmio mais alto. Este grafo é apresentado na Figura 7.1.
- **Insight de Explicabilidade:** Esta etapa representa um avanço na explicabilidade do processo. O sistema não apenas selecionou fatos, mas construiu uma hipótese estrutural de como eles se conectam. Nesse processo, novos nós são recuperados para a formação de um subgrafo coeso. O subgrafo resultante constitui a saída da Fase I do pipeline executado pelo framework ARANDU.

Fase II: Orquestração Lógica do Contexto

Após a recuperação de um subgrafo estruturalmente coeso, o pipeline do ARANDU avança para a Fase II. O objetivo desta etapa é enriquecer o subgrafo e ponderar seus caminhos por meio de raciocínio simbólico explícito. Este processo é fundamental

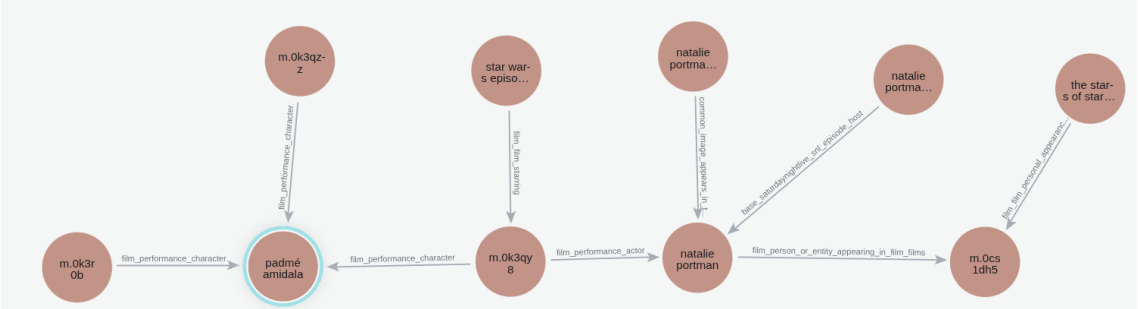

Figura 7.1: Subgrafo coeso resultante da fase de recuperação estruturada

para garantir a completude lógica e a relevância contextual da informação que será, subsequentemente, apresentada ao LLM. Como visto no Capítulo 4, a fase é executada em dois módulos sequenciais: o Enriquecimento Lógico e a Ponderação de Caminhos.

1. Módulo de Enriquecimento Lógico (LogicalEnricher)

- **Entrada:** O subgrafo coeso recuperado pela Fase I (Figura 7.1).
- **Processo:** O `LogicalEnricher` inspeciona o subgrafo em busca de lacunas inferenciais, aplicando regras lógicas previamente mineradas (conforme Seção 5.4.3) para materializar conhecimento implícito. Para a consulta em questão, quatro regras apresentam elementos (nós ou arestas) existentes no subgrafo. A Tabela 7.16 apresenta as regras aplicadas. Como visto no Capítulo 5, para aplicação das regras o subgrafo é convertido em um grafo RDF. A Tabela 7.17 apresenta o grafo antes e depois do enriquecimento representado através de triplas RDF.
- **Saída (Subgrafo Enriquecido):** A Figura 7.2 apresenta uma representação visual do grafo enriquecido. Em comparação com a Figura 7.1, o grafo enriquecido contém cinco novas triplas (arestas na figura) inferidas.
- **Insight de Explicabilidade:** Esta etapa demonstra a capacidade do ARANDU de ir além da simples recuperação de fatos existentes. O sistema realiza inferência lógica para completar o contexto. Esta inferência é totalmente rastreável à regra aplicada. Na Figura 7.2 é possível observar que, após a adição das triplas inferidas, o grafo enriquecido contém um caminho explícito e contínuo da atriz ao personagem. A explicabilidade é aprimorada, pois o raciocínio que leva do ator ao personagem é agora justificado por uma regra formal do domínio.

2. Módulo de Ponderação de Caminhos (PathRanker)

- **Entrada:** O subgrafo enriquecido pelo módulo anterior.
- **Processo:** O `PathRanker` procura pontuar caminhos que são mais relevantes para a pergunta. Para isso, ele utiliza as evidências retornadas pelo `TextRetriever`,

Tabela 7.16: Regras lógicas aplicadas durante o enriquecimento do subgrafo

Corpo da Regra	**Cabeça da Regra**
`?b film_film_starring ?a`	`?a film_performance_film ?b`
`?b film_performance_actor ?a`	`?a film_actor_film ?b`
`?b film_actor_film ?a`	`?a film_performance_actor ?b`
`?b film_performance_character ?a`	`?a film_film_character_portrayed_in_films ?b`

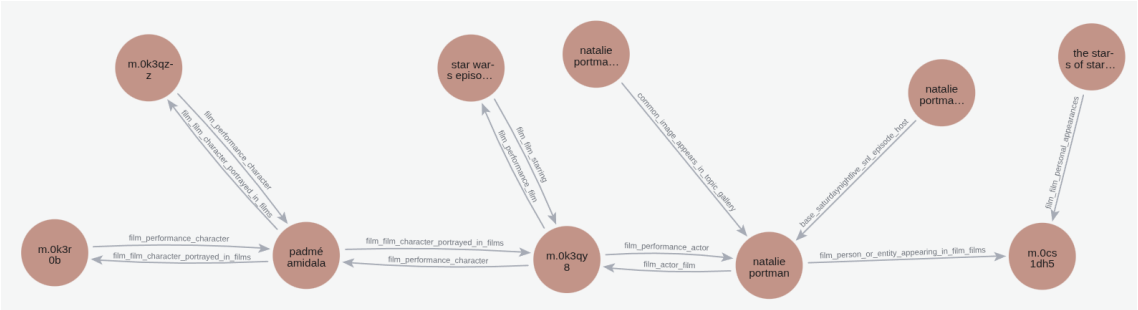

Figura 7.2: Subgrafo enriquecido após aplicação das regras lógicas

Tabela 7.17: Comparação entre o subgrafo original e o subgrafo enriquecido logicamente

Subgrafo Original	**Subgrafo Enriquecido**

`@prefix ex: <http://example.org/> .`
`ex:748 ex:film_performance_character ex:480 .`
`ex:766 ex:film_performance_character ex:480 .`
`ex:479 ex:film_performance_character ex:480 .`
`ex:1094 ex:film_film_starring ex:748 .`
`ex:1227 ex:film_film_personal_appearances ex:667 .`
`ex:33 ex:base_saturdaynightlive_snl_episode_host ex:162 .`
`ex:647 ex:common_image_appears_in_topic_gallery ex:162 .`
`ex:162 ex:film_person_or_entity_appearing_in_film_films ex:667 .`
`ex:748 ex:film_performance_actor ex:162 .`
	`@prefix ex: <http://example.org/> .`
	`ex:480 ex:film_film_character_portrayed_in_films ex:479 , ex:766 ,`
	`↪ex:748 .`
	`ex:162 ex:film_actor_film ex:748 .`
	`ex:748 ex:film_performance_film ex:1094 ;`
	`ex:film_performance_character ex:480 .`
	`ex:766 ex:film_performance_character ex:480 .`
	`ex:479 ex:film_performance_character ex:480 .`
	`ex:1094 ex:film_film_starring ex:748 .`
	`ex:1227 ex:film_film_personal_appearances ex:667 .`
	`ex:33 ex:base_saturdaynightlive_snl_episode_host ex:162 .`
	`ex:647 ex:common_image_appears_in_topic_gallery ex:162 .`
	`ex:162 ex:film_person_or_entity_appearing_in_film_films ex:667 .`
	`ex:748 ex:film_performance_actor ex:162 .`

ainda na Fase I, e atribui pontos aos caminhos de até 2 saltos. Assim, caminhos com nós como "Natalie Portman"e arestas como "film_performance_character"são pontuados. Ao final desse processo, é extraída uma lista de caminhos ordenada por pontuação, conforme apresentado na Tabela 7.18. Os nós e arestas são então ponderados de acordo com a pontuação dos caminhos que elas compõem.

- **Saída (Subgrafo Ponderado):** O módulo retorna o mesmo subgrafo, mas com um novo atributo de peso (`score`) associado a cada aresta e nó. As arestas que compõem os caminhos de maior pontuação recebem pesos mais elevados. As Tabelas 7.19 e 7.20 apresentam os nós e arestas ordenados por pontuação, após esse processo.
- **Insight de Explicabilidade:** A ponderação atua como um mecanismo de **atenção simbólica**. Antes mesmo de envolver o LLM, o ARANDU já priorizou a cadeia de evidências mais lógica e relevante. Isso não apenas foca a etapa de geração,

mas também permite que o sistema explique por que certas evidências foram consideradas mais importantes que outras, baseando a justificativa nos pesos de relevância dos caminhos. Na Figura 7.3 o caminho de maior pontuação é destacado, mostrando que o sistema priorizou o caminho que leva da atriz ao personagem. A partir dos resultados apresentados nas Tabelas 7.18, 7.19 e 7.20 é possível observar que a evidência para resposta correta e priorizada e está logo no início da lista (top 1).

Tabela 7.18: Caminhos ponderados pelo módulo PathRanker

Pontuação	**Caminho (Triplas)**
1.40	`(480, film_film_character_portrayed_in_films, 748),` `(748, film_performance_actor, 162)`
1.20	`(1094, film_film_starring, 748),` `(748, film_performance_character, 480)`
1.00	`(162, film_actor_film, 748),` `(748, film_performance_character, 480)`
0.80	`(480, film_film_character_portrayed_in_films, 748),` `(748, film_performance_film, 1094)`
0.80	`(1094, film_film_starring, 748),` `(748, film_performance_actor, 162)`
0.60	`(748, film_performance_actor, 162)`
0.40	`(162, film_person_or_entity_appearing_in_film_films, 667)`
0.20	`(1094, film_film_starring, 748)`

Tabela 7.19: Nós ponderados pelo módulo PathRanker

ID do Nó	**Texto do Nó**	**Pontuação**
480	padmé amidala	4.4
162	natalie portman	4.2
1094	star wars episode i: the phantom menace	3.0
748	m.0k3qy8	0.8
667	m.0cs1dh5	0.4
33	natalie portman/fall out boy	0.0
479	m.0k3qzz	0.0
647	natalie portman at tiff 2009	0.0
766	m.0k3r0b	0.0
1227	the stars of star wars	0.0

Fase III: Representação Neural e Geração da Resposta

Na fase final do pipeline, o objetivo é traduzir o subgrafo de evidências, já enriquecido e ponderado, em um formato que preserve sua estrutura e guie o modelo de linguagem a uma resposta fiel e precisa. Como visto no Capítulo 4, a fase é executada em dois módulos sequenciais: o Módulo de Representação Neural via GNN e o Módulo de Geração Textual. Na implementação do framework, os dois módulos são integrados através de um único componente: o `SoftPromptGraphModel`.

Tabela 7.20: Arestas ponderadas pelo módulo PathRanker

Nó Origem	Nó Destino	Tipo da Relação	Pontuação
748	162	film_performance_actor	2.2
480	748	film_film_character_portrayed_in_films	1.4
162	748	film_actor_film	1.0
748	480	film_performance_character	1.0
1094	748	film_film_starring	0.8
162	667	film_person_or_entity_appearing_in_film_films	0.8
33	162	base_saturdaynightlive_snl_episode_host	0.4
647	162	common_image_appears_in_topic_gallery	0.4
480	479	film_film_character_portrayed_in_films	0.0
480	766	film_film_character_portrayed_in_films	0.0
748	1094	film_performance_film	0.0
766	480	film_performance_character	0.0
479	480	film_performance_character	0.0
1227	667	film_film_personal_appearances	0.0

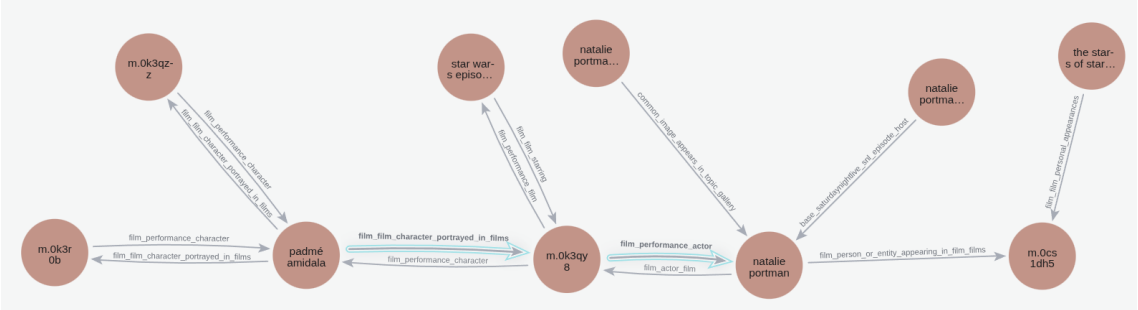

Figura 7.3: Caminho melhor pontuado em destaque no subgrafo ponderado

1. Módulo de Representação Neural via GNN

- **Entrada:** O subgrafo enriquecido e ponderado proveniente da Fase II. As arestas deste grafo contêm os pesos de relevância atribuídos pelo `PathRanker`.
- **Processo:** O subgrafo é processado por uma Rede Neural de Grafos (GNN), especificamente uma *Graph Attention Network v2* (GATv2) [Brody, Alon e Yahav 2022]. A GATv2 representa uma evolução da arquitetura GAT original, oferecendo maior expressividade através de um mecanismo de atenção dinâmica que permite diferentes pesos de atenção para diferentes camadas da rede. A GATv2 utiliza os pesos das arestas para modular seus mecanismos de atenção, focando nos caminhos de maior relevância lógica. O modelo agrega as informações dos nós e das relações em uma representação vetorial única e densa, que encapsula tanto a semântica quanto a topologia priorizada do subgrafo.
- **Saída (*Graph Token*):** O módulo produz um único vetor de alta dimensão, o `graph token`. Este vetor funciona como um *soft prompt*, uma instrução contínua que será injetada diretamente no espaço de embeddings do LLM.
- **Insight de Explicabilidade:** Esta etapa é importante para mitigar o problema da perda de informação na linearização textual, uma limitação de muitos sistemas

`GraphRAG` [Pan et al. 2024]. O `graph token` fornece um viés indutivo estrutural. Ele instrui o LLM a raciocinar de uma forma que seja consistente com a estrutura lógica do conhecimento validado, traduzindo a relevância simbólica em um sinal processável pela rede neural.

2. Módulo de Geração Textual

- **Entrada:** Um prompt híbrido composto por três elementos:
  1. A pergunta original do usuário.
  2. O *graph token* gerado pela GNN.
  3. A representação textual (linearização) dos nós e arestas do subgrafo, ordenadas pelos pesos de relevância. Essa ordenação é importante por dois motivos. Primeiro, o limite textual da janela de contexto do LLM. Segundo, para mitigar o efeito de "perdido no meio" (*lost in the middle*) [Liu et al. 2023]. Esse efento mostra que o LLM tende a dar mais atenção às informações presentes no início e no fim do contexto, ignorando dados importantes que aparecem no meio.

  Como apresentado no Capítulo 5, para composição do prompt é utilizado um *chat template*. A Figura 7.4 apresenta o prompt construido para a pergunta selecionada.
- **Processo:** O prompt é submetido ao LLM de base (e.g., Llama 3.1 8B). Como é possível observar na Figura 7.4, é utilizada uma role específica (`graph`) e um token especial (`<GRAPH_TOKEN>`) no prompt. Esse *placeholder* é substituido pelo *graph token* gerado pela GNN em nível de embeddings, antes das camadas de atenção do LLM. Este, por sua vez, utiliza o contexto de entrada para sintetizar a resposta final.
- **Saída (Resposta Final):** O módulo gera a resposta em linguagem natural. Para a nossa consulta exemplo, a saída gerada é: `padmé amidala`, correspondente à resposta correta.
- **Insight de Explicabilidade (Rastreabilidade de Ponta a Ponta):** A resposta gerada é diretamente rastreável à cadeia de evidências construída e validada ao longo de todo o pipeline. A menção a "Padmé Amidala"não é uma "alucinação"do LLM, mas uma consequência direta da recuperação e priorização dos caminhos mais relevantes. O sistema pode, portanto, não apenas fornecer a resposta, mas também apresentar o subgrafo ponderado como a justificativa explícita para sua conclusão.

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 12 Aug 2025

<|eot_id|><|start_header_id|>graph<|end_header_id|>
<GRAPH_TOKEN>
<|eot_id|><|start_header_id|>user<|end_header_id|>

node_id,node_attr
480,padmé amidala
162,natalie portman
1094,star wars episode i: the phantom menace
748,m.0k3qy8
667,m.0cs1dh5
33,natalie portman/fall out boy
479,m.0k3qzz
647,natalie portman at tiff 2009
766,m.0k3r0b
1227,the stars of star wars

src,edge_attr,dst
480,film_film_character_portrayed_in_films,748
748,film_performance_character,480
748,film_performance_actor,162
1094,film_film_starring,748
162,film_actor_film,748
748,film_performance_film,1094
480,film_film_character_portrayed_in_films,479

Please answer the given question without any explanation.
Question: what character did natalie portman play in star wars?
Answer:<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Figura 7.4: Prompt construído para a consulta exemplo, contendo o *chat template* do Llama 3.1, o *placeholder* do *graph token* (`<GRAPH_TOKEN>`) e a linearização do subgrafo ponderado

CAPÍTULO 8

Considerações Finais

A proliferação de Grandes Modelos de Linguagem (LLMs) transformou o campo do Processamento de Linguagem Natural, mas também expôs suas limitações intrínsecas. Notadamente a propensão a gerar informações factualmente inconsistentes ou desatualizadas. O paradigma de Geração Aumentada por Recuperação (RAG) emergiu como uma solução robusta, mas sua eficácia é condicionada pela qualidade das evidências recuperadas. Conforme discutido no Capítulo 1, a utilização de Grafos de Conhecimento como fonte de evidências representa uma evolução natural, porém introduz novos desafios relacionados à extração de subgrafos que sejam ao mesmo tempo relevantes, coesos e logicamente consistentes. Diante desse cenário, esta tese partiu da hipótese de que a fidelidade e a precisão de sistemas de Resposta a Perguntas sobre Grafos de Conhecimento poderiam ser significativamente aprimoradas por meio de uma arquitetura neuro-simbólica. Tal arquitetura deveria ser capaz de aliar a eficiência das abordagens de recuperação neural com o rigor da validação lógica. Para investigar essa premissa, este trabalho percorreu uma trajetória que se iniciou com a fundamentação teórica dos conceitos-chave (Capítulo 2), passando pela proposição do *framework* ARANDU (Capítulo 4) e sua validação empírica frente a baselines do estado da arte (Capítulo 7).

Este capítulo final consolida a investigação realizada. A primeira seção detalha as principais contribuições científicas e tecnológicas decorrentes da pesquisa. A segunda seção, por sua vez, discute de forma transparente as limitações inerentes ao trabalho e, a partir delas, delineia um conjunto de direções promissoras para trabalhos futuros.

8.1 Conclusões de Pesquisa

Conforme definido no Capítulo 1, o problema central desta tese foi superar a tripla limitação dos sistemas KGQA existentes, relacionada aos desafios de recuperação, representação e orquestração. A hipótese central postulou que uma arquitetura neuro-simbólica integrada poderia apoiar na resolução desses desafios de forma sinérgica. A validação empírica do *framework* ARANDU, detalhada no Capítulo 7, forneceu as evidências para confirmar esta hipótese, permitindo extrair as seguintes conclusões.

A investigação sobre o desempenho geral (QP1) demonstrou que a abordagem híbrida de recuperação, combinada com a otimização estrutural, ataca diretamente o problema da baixa precisão e cobertura. A superioridade do ARANDU sobre os *baselines* valida a primeira parte da hipótese, confirmando que a recuperação semântica e estrutural, juntas, são mais eficazes do que isoladamente.

A análise do componente de orquestração lógica (QP2) demonstrou que essa fase é importante para a qualidade da recuperação de evidências. Os resultados quantitativos revelaram que o ARANDU supera significativamente os baselines em métricas de qualidade de ranqueamento (nDCG@K e MRR), especialmente em cenários de maior complexidade. A superioridade do ARANDU em nDCG@10 (70,22% versus 47,68% do G-Retriever no WebQSP) e MRR (49,66% versus 24,00% no MetaQA 2-hop) evidencia que a orquestração lógica produz subgrafos mais coerentes e fieis à pergunta. Esta capacidade de priorizar evidências relevantes nas primeiras posições do ranqueamento estabelece uma base sólida para a geração de respostas de alta qualidade, validando a segunda parte da hipótese.

A investigação sobre a representação neural (QP3) demonstrou que a operacionalização estrutural via GNN impacta positivamente a qualidade da resposta final. O estudo de ablação revelou melhorias em todas as métricas avaliadas, com destaque para a precisão (de 29,92% para 57,06%) e F1-Score (de 37,59% para 53,55%). A comparação entre ARANDU completo e ARANDU sem GNN evidenciou que a codificação neural preserva informações topológicas que se perdem na linearização textual, resultando em respostas mais objetivas e focadas. Isso valida a terceira parte da hipótese, confirmando que a representação neural estrutural contribui significativamente para a qualidade da geração final.

A análise de eficiência computacional (QP4) confirmou que o ARANDU mantém uma eficiência computacional compatível com seu princípio de design de ser uma solução leve. Os resultados demonstraram latência competitiva de 1036ms, posicionando-se próximo ao GraphRAG (953ms) e significativamente abaixo do G-Retriever (7212ms). A arquitetura híbrida permite otimização de recursos, com o LLM concentrando o uso de GPU (95%) e VRAM (6650MB), enquanto módulos como PCST e orquestração lógica utilizam predominantemente CPU. A escalabilidade se mostrou robusta, com tempo de resposta estável mesmo com o crescimento do grafo, evidenciando que o investimento em processamento offline resulta em eficiência operacional online.

Em suma, a investigação conduzida nesta tese confirma a hipótese central. A sinergia entre os componentes de recuperação, orquestração e representação, materializada na arquitetura neuro-simbólica do ARANDU, constitui uma solução robusta e eficaz para os desafios contemporâneos de KGQA.

8.2 Limitações e Trabalhos Futuros

Toda investigação científica possui fronteiras e suposições que definem seu escopo. É fundamental reconhecer as limitações do presente trabalho para contextualizar suas contribuições e, principalmente, para delinear caminhos promissores para pesquisas futuras. Esta seção discute as principais limitações da arquitetura ARANDU e propõe direções de pesquisa que emergem a partir delas.

8.2.1 Limitações do Estudo

Ainda que a representação de contexto através de grafos de conhecimento seja uma abordagem que tem ganhado cada vez mais popularidade, a construção de novos grafos de conhecimento de alta qualidade é um desafio significativo. A principal limitação desta tese reside na **dependência de um Grafo de Conhecimento pré-existente e de alta qualidade**. O *framework* ARANDU foi projetado para operar sobre GCs já construídos, assumindo que o conhecimento está corretamente estruturado e é factualmente acurado. A sua eficácia, portanto, é diretamente condicionada pela qualidade e cobertura da base de conhecimento subjacente. O processo de construção do Grafo de Conhecimento em si não foi o foco deste trabalho.

Uma segunda limitação está relacionada à **natureza estática do componente de raciocínio lógico**. As regras de Horn são mineradas em uma etapa offline e permanecem fixas durante a execução. Essa abordagem se mostrou eficaz e, pela sua velocidade de execução, pode ser executada periodicamente, a medida que o grafo de conhecimento é atualizado. Ainda assim, há a limitação da inferência de novas regras em tempo real. O raciocínio é puramente dedutivo, baseado em regras pré-estabelecidas.

Uma terceira limitação relevante refere-se à **ausência de mecanismos de retroalimentação entre os componentes da arquitetura**. O *framework* ARANDU foi projetado para execução de um *pipeline* unidirecional, onde cada componente opera de forma sequencial sem comunicação reversa. A camada de recuperação textual não se beneficia dos resultados da orquestração lógica, e o componente de representação neural não influencia a estratégia de recuperação de subgrafos. Esta foi uma escolha arquitetural intencional, visando o desempenho e redução da latência na resposta a uma pergunta. Todavia, tal limitação pode resultar em oportunidades perdidas de otimização iterativa, onde informações derivadas de etapas posteriores poderiam refinar o processamento de etapas anteriores.

8.2.2 Trabalhos Futuros

As limitações identificadas abrem um leque de oportunidades para futuras investigações. Propomos as seguintes direções de pesquisa:

- **Construção e Enriquecimento Automático de GCs:** Uma extensão natural do trabalho seria integrar o ARANDU a sistemas de Extração de Informação. Isso permitiria a construção e atualização automática do GC a partir de documentos textuais não estruturados. Tal integração resultaria em um sistema de ponta a ponta, capaz de transformar corpus textuais em bases de conhecimento consultáveis e de alta fidelidade.
- **Aprendizado Dinâmico de Regras Lógicas:** Para superar a natureza estática das regras, futuras pesquisas poderiam explorar técnicas de Programação Lógica Indutiva (ILP) ou métodos híbridos de mineração de padrões em tempo real. Sistemas de *neural rule induction* baseados em representações simbólicas poderiam aprender a descobrir e validar novas regras de Horn dinamicamente, preservando a interpretabilidade do componente lógico. Essa abordagem conferiria maior adaptabilidade ao sistema sem comprometer a natureza neuro-simbólica da arquitetura.
- **Integração de Ontologias na Orquestração Lógica:** Por utilizar SPARQL, a fase de orquestração lógica do ARANDU permite a representação do grafo através de padrões da Web Semântica (como OWL e RDF). Essa abordagem poderia ser aprimorada através da incorporação de ontologias formais como base para o raciocínio. Ontologias estruturadas em linguagens como OWL poderiam fornecer um *framework* semântico mais rico para a validação e refinamento de subgrafos. Isso permitiria a utilização de axiomas ontológicos, restrições de domínio e hierarquias conceituais para guiar o processo de orquestração, resultando em uma validação lógica mais sofisticada.
- **Extensão para Grafos de Conhecimento Multimodais:** O conhecimento humano é inerentemente multimodal. Uma direção de pesquisa promissora é adaptar a arquitetura para operar sobre GCs que integrem diferentes modalidades de dados, como imagens, tabelas e textos. Isso exigiria o desenvolvimento de novos mecanismos de recuperação e representação capazes de fundir informações de múltiplas fontes. A arquitetura modular do ARANDU é facilmente expandível para incluir novos componentes, capazes de operar sobre múltiplas modalidades de dados.
- **Explicabilidade Aprimorada e Auditabilidade:** O componente simbólico do ARANDU gera um traço de raciocínio explícito. Uma linha de trabalho futura de grande valor prático seria desenvolver módulos capazes de traduzir esse traço em explicações em linguagem natural. Isso tornaria o processo de tomada de decisão do sistema transparente e auditável por usuários finais.

- **Resposta a Perguntas em Cenários Conversacionais:** A presente pesquisa focou em interações de um único turno. Estender o *framework* para cenários de QA conversacional é um passo seguinte lógico. Isso demandaria a implementação de mecanismos de gerenciamento de estado de diálogo e resolução de co-referências para manter o contexto ao longo de múltiplas interações.

Referências Bibliográficas

[Abdin et al. 2024]ABDIN, M. et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024. Disponível em: <https://arxiv.org/abs/2412.08905>.

[Anand 2020]ANAND, R. An illustrated guide to graph neural networks. *Rishabh Anand Medium Blog*, 2020.

[Angles et al. 2018]ANGLES, R. et al. G-core: A core for future graph query languages. *Journal of Computer and System Sciences*, Elsevier, v. 104, p. 318–335, 2018.

[Baader et al. 2003]BAADER, F. et al. *The description logic handbook: Theory, implementation and applications*. [S.l.]: Cambridge university press, 2003.

[Bahdanau, Cho e Bengio 2016]BAHDANAU, D.; CHO, K.; BENGIO, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. Disponível em: <https://arxiv.org/abs/1409.0473>.

[Bengio et al. 2003]BENGIO, Y. et al. A neural probabilistic language model. *Journal of machine learning research*, v. 3, n. Feb, p. 1137–1155, 2003.

[Berant et al. 2013]BERANT, J. et al. Semantic parsing on freebase from question-answer pairs. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. [S.l.: s.n.], 2013. p. 1533–1544.

[Bizer et al. 2007]BIZER, C. et al. Dbpedia: A nucleus for a web of open data. In: *Proceedings of the 6th International Semantic Web Conference (ISWC)*. [S.l.]: Springer, 2007. p. 722–735.

[Bollacker et al. 2008]BOLLACKER, K. et al. Freebase: a collaboratively created graph database for structuring human knowledge. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. [S.l.: s.n.], 2008. p. 1247–1250.

[Bordes et al. 2013]BORDES, A. et al. Translating embeddings for modeling multi-relational data. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2013. v. 26.

[Brachman e Levesque 2004]BRACHMAN, R. J.; LEVESQUE, H. J. *Knowledge Representation and Reasoning*. [S.l.]: Morgan Kaufmann, 2004.

[Brickley e Miller 2014]BRICKLEY, D.; MILLER, L. *FOAF Vocabulary Specification 0.99*. [S.l.]: W3C, January 2014. http://xmlns.com/foaf/spec/.

[Brody, Alon e Yahav 2022]BRODY, S.; ALON, U.; YAHAV, E. How attentive are graph attention networks? In: *International Conference on Learning Representations (ICLR)*. [s.n.], 2022. Disponível em: <https://openreview.net/forum?id=P1sttV-TCf>.

[Brown et al. 2020]BROWN, T. et al. Language models are few-shot learners. In: LAROCHELLE, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. v. 33, p. 1877–1901. Disponível em: <https://arxiv.org/abs/2005.14165>.

[Brown et al. 2020]BROWN, T. et al. Language models are few-shot learners. *Advances in neural information processing systems*, v. 33, p. 1877–1901, 2020.

[Cao et al. 2025]CAO, R. et al. Neusym-rag: Hybrid neural symbolic retrieval with multiview structuring for pdf question answering. *arXiv preprint arXiv:2505.19754*, 2025. Disponível em: <https://arxiv.org/abs/2505.19754>.

[Carpineto e Romano 2012]CARPINETO, C.; ROMANO, G. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys*, v. 44, n. 1, p. 1–50, 2012.

[Chen et al. 2025]CHEN, B. et al. Pathrag: Pruning graph-based retrieval augmented generation with relational paths. *arXiv preprint arXiv:2502.14902*, 2025. Disponível em: <https://arxiv.org/abs/2502.14902>.

[Chen et al. 2024]CHEN, Z. et al. Rulerag: Rule-guided retrieval-augmented generation with language models for question answering. *arXiv preprint arXiv:2410.22353*, 2024. Disponível em: <https://arxiv.org/abs/2410.22353>.

[CHROMA 2025]CHROMA. *Chroma DB: A Database for Embeddings and Vector Search*. 2025. https://www.trychroma.com/. [online; visited on 2025-05-14].

[Consortium et al. 2014]CONSORTIUM, W. W. W. et al. Rdf 1.1 primer. World Wide Web Consortium, 2014.

[Cormack, Clarke e Buettcher 2009]CORMACK, G. V.; CLARKE, C. L.; BUETTCHER, S. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. [S.l.: s.n.], 2009. p. 758–759.

[Dao 2023]DAO, T. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. 2023. Disponível em: <https://arxiv.org/abs/2307.08691>.

[Dettmers et al. 2023]DETTMERS, T. et al. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023. Disponível em: <https://arxiv.org/abs/2305.14314>.

[Devlin et al. 2019]DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423>.

[Diestel 2017]DIESTEL, R. *Graph Theory*. 5. ed. [S.l.]: Springer, 2017. (Graduate Texts in Mathematics, v. 173).

[Edge et al. 2024]EDGE, D. et al. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. 2024. Disponível em: <https://arxiv.org/abs/2404.16130>.

[Elman 1990]ELMAN, J. L. Finding structure in time. *Cognitive science*, Wiley Online Library, v. 14, n. 2, p. 179–211, 1990.

[Fey e Lenssen 2019]FEY, M.; LENSSEN, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. Disponível em: <https://arxiv.org/abs/1903.02428>.

[Furnas et al. 1987]FURNAS, G. W. et al. The vocabulary problem in human-system communication. *Communications of the ACM*, v. 30, n. 11, p. 964–971, 1987.

[Gagniuc 2017]GAGNIUC, P. A. *Markov chains: from theory to implementation and experimentation*. [S.l.]: John Wiley & Sons, 2017.

[Gao et al. 2020]GAO, L. et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[Gao et al. 2023]GAO, Y. et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023. Disponível em: <https://arxiv.org/abs/2312.10997>.

[Goemans e Williamson 1995]GOEMANS, M. X.; WILLIAMSON, D. P. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, SIAM, v. 24, n. 2, p. 296–317, 1995.

[Group 2012]GROUP, W. O. W. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. [S.l.]: W3C Recommendation, December 2012. https://www.w3.org/TR/owl2-overview/.

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

[Guo et al. 2025]GUO, Z. et al. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2025. Disponível em: <https://arxiv.org/abs/2410.05779>.

[Hamilton, Ying e Leskovec 2018]HAMILTON, W. L.; YING, R.; LESKOVEC, J. *Inductive Representation Learning on Large Graphs*. 2018. Disponível em: <https://arxiv.org/abs/1706.02216>.

[Han et al. 2025]HAN, H. et al. Retrieval-augmented generation with graphs (graphrag). *CoRR*, abs/2501.00309, January 2025. Disponível em: <https://doi.org/10.48550/arXiv.2501.00309>.

[Harris e Seaborne 2013]HARRIS, S.; SEABORNE, A. *SPARQL 1.1 Query Language*. [S.l.]: W3C Recommendation, March 2013. https://www.w3.org/TR/sparql11-query/.

[Haugeland 1985]HAUGELAND, J. *Artificial intelligence: The very idea*. [S.l.]: MIT press, 1985.

[He et al. 2024]HE, X. et al. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. In: *Advances in Neural Information Processing Systems*. [s.n.], 2024. v. 37, p. 132876–132907. Disponível em: <https://arxiv.org/abs/2402.07630>.

[Hochreiter e Schmidhuber 1997]HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT press, v. 9, n. 8, p. 1735–1780, 1997.

[Hogan et al. 2021]HOGAN, A. et al. Knowledge graphs. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 8, oct 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3447772>.

[Huang et al. 2025]HUANG, L. et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, Association for Computing Machinery, New York, NY, USA, v. 43, n. 2, jan. 2025. ISSN 1046-8188. Disponível em: <https://doi.org/10.1145/3703155>.

[ISO/IEC 2024]ISO/IEC. *ISO/IEC 39075:2024 — Information technology — Database languages — GQL*. Geneva, Switzerland, abr. 2024. Disponível em: <https://www.iso.org/standard/76120.html>.

[Ji et al. 2023]JI, Z. et al. A survey of hallucination in natural language generation. *ACM Computing Surveys*, v. 55, n. 12, p. 1–38, 2023. Disponível em: <https://doi.org/10.1145/3571730>.

[Johnson, Douze e Jégou 2019]JOHNSON, J.; DOUZE, M.; JÉGOU, H. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, IEEE, v. 7, n. 3, p. 535–547, 2019.

[Jurafsky e Martin 2023]JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 3rd. ed. Prentice Hall, 2023. Draft of December 29, 2023. Disponível em: <https://web.stanford.edu/ jurafsky/slp3/>.

[Kang et al. 2023]KANG, M. et al. *Knowledge Graph-Augmented Language Models for Knowledge-Grounded Dialogue Generation*. 2023. Disponível em: <https://arxiv.org/abs/2305.18846>.

[Kaplan et al. 2020]KAPLAN, J. et al. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[Karpukhin et al. 2020]KARPUKHIN, V. et al. Dense passage retrieval for open-domain question answering. In: *Proceedings of EMNLP 2020*. [S.l.: s.n.], 2020. p. 6769–6781.

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

[Kipf e Welling 2017]KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. In: *International Conference on Learning Representations (ICLR)*. [s.n.], 2017. Disponível em: <https://openreview.net/forum?id=SJU4ayYgl>.

[Lajus, Galárraga e Suchanek 2020]LAJUS, J.; GALÁRRAGA, L.; SUCHANEK, F. M. Fast and exact rule mining with AMIE 3. In: *The Semantic Web – ESWC 2020*. Springer, 2020. p. 36–52. Disponível em: <https://doi.org/10.1007/978-3-030-49461-2_3>.

[Lester, Al-Rfou e Constant 2021]LESTER, B.; AL-RFOU, R.; CONSTANT, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021. Disponível em: <https://arxiv.org/abs/2104.08691>.

[Lewis et al. 2020]LEWIS, P. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: *Advances in Neural Information Processing Systems*. [s.n.], 2020. v. 33, p. 9459–9474. Disponível em:

$<$https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf$>$.

[Li et al. 2023]LI, J. et al. *BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models*. 2023. Disponível em: $<$https://arxiv.org/abs/2301.12597$>$.

[Li et al. 2023]LI, X. et al. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. *arXiv preprint arXiv:2305.13269*, 2023.

[Li e Liang 2021]LI, X. L.; LIANG, P. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. Disponível em: $<$https://arxiv.org/abs/2101.00190$>$.

[Li et al. 2015]LI, Y. et al. Gated graph sequence neural networks. In: *Proceedings of ICLR'16*. [s.n.], 2015. Disponível em: $<$https://arxiv.org/abs/1511.05493$>$.

[Lin et al. 2015]LIN, Y. et al. Learning entity and relation embeddings for knowledge graph completion. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2015. v. 29, n. 1.

[Liu et al. 2023]LIU, N. F. et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. Disponível em: $<$https://arxiv.org/abs/2307.03172$>$.

[Loshchilov e Hutter 2019]LOSHCHILOV, I.; HUTTER, F. Decoupled weight decay regularization. In: *International Conference on Learning Representations*. [s.n.], 2019. Disponível em: $<$https://openreview.net/forum?id=Bkg6RiCqY7$>$.

[Luo et al. 2024]LUO, L. et al. *Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning*. 2024. Disponível em: $<$https://arxiv.org/abs/2310.01061$>$.

[Lù 2024]Lù, X. H. *BM25S: Orders of magnitude faster lexical search via eager sparse scoring*. 2024. Disponível em: $<$https://arxiv.org/abs/2407.03618$>$.

[Ma et al. 2025]MA, S. et al. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In: *The Thirteenth International Conference on Learning Representations (ICLR)*. [s.n.], 2025. Disponível em: $<$https://arxiv.org/abs/2407.10805$>$.

[Maliakel, Ilager e Brandic 2025]MALIAKEL, P. J.; ILAGER, S.; BRANDIC, I. Investigating energy efficiency and performance trade-offs in llm inference across tasks and dvfs settings. *arXiv preprint arXiv:2501.08219*, 2025. Disponível em: $<$https://arxiv.org/abs/2501.08219$>$.

[Malkov e Yashunin 2020]MALKOV, Y. A.; YASHUNIN, D. A. Efficient and robust approximate nearest neighbor search using hnsw. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 42, n. 4, p. 824–836, 2020.

[Manning, Raghavan e Schütze 2008]MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008. ISBN 9780521865715.

[Miller et al. 2016]MILLER, A. H. et al. Key-value memory networks for directly reading documents. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Austin, TX: Association for Computational Linguistics, 2016. p. 1400–1409. Disponível em: <https://aclanthology.org/D16-1147>.

[Muja e Lowe 2014]MUJA, M.; LOWE, D. G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 36, n. 11, p. 2227–2240, 2014.

[Neo4j Inc. 2025]Neo4j Inc. *Cypher Query Language*. [S.l.], 2025. Acesso em: 22 ago. 2025. Disponível em: <https://neo4j.com/developer/cypher/>.

[Neo4j, Inc. 2025]Neo4j, Inc. *Neo4j Graph Database*. 2025. https://neo4j.com/. Acesso em: 31 jul. 2025.

[Ng, Matsuba e Zhang 2025]NG, K. K. Y.; MATSUBA, I.; ZHANG, P. C. Rag in health care: a novel framework for improving communication and decision-making by addressing llm limitations. *Nejm Ai*, Massachusetts Medical Society, v. 2, n. 1, p. AIra2400380, 2025.

[NVIDIA 2023]NVIDIA. *Post-Training Quantization of LLMs with NVIDIA NeMo and TensorRT-LLM*. 2023. https://developer.nvidia.com/blog/. Acesso em: 22 ago. 2025.

[Ouyang et al. 2022]OUYANG, L. et al. Training language models to follow instructions with human feedback. In: *Advances in Neural Information Processing Systems*. [S.l.]: Curran Associates, Inc., 2022. v. 35, p. 27730–27744.

[Pan et al. 2024]PAN, S. et al. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 36, n. 7, p. 3580–3599, jul. 2024. ISSN 2326-3865. Disponível em: <http://dx.doi.org/10.1109/TKDE.2024.3352100>.

[Paszke et al. 2019]PASZKE, A. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. Disponível em: <https://arxiv.org/abs/1912.01703>.

[Peng et al. 2024]PENG, B. et al. *Graph Retrieval-Augmented Generation: A Survey*. 2024. Disponível em: <https://arxiv.org/abs/2408.08921>.

[Plenz e Frank 2024]PLENZ, M.; FRANK, A. Graph language models. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, 2024. p. 4477–4494. Disponível em: <https://aclanthology.org/2024.acl-long.245/>.

[Plenz e Frank 2024]PLENZ, M.; FRANK, A. Graph language models. *arXiv preprint arXiv:2401.07105*, 2024.

[Prud'hommeaux e Seaborne 2008]PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF*. [S.l.], jan. 2008. Disponível em: <https://www.w3.org/TR/rdf-sparql-query/>.

[Rafailov et al. 2023]RAFAILOV, R. et al. Direct preference optimization: Your language model is secretly a reward model. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2023. v. 36, p. 34994–35012.

[Raffel et al. 2020]RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, v. 21, n. 140, p. 1–67, 2020.

[Rashkin et al. 2023]RASHKIN, H. et al. Measuring attribution in natural language generation models. *Computational Linguistics*, MIT Press, Cambridge, MA, v. 49, n. 4, p. 777–840, dez. 2023. Disponível em: <https://aclanthology.org/2023.cl-4.2/>.

[Reimers e Gurevych 2019]REIMERS, N.; GUREVYCH, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. Disponível em: <https://arxiv.org/abs/1908.10084>.

[Reimers e Gurevych 2020]REIMERS, N.; GUREVYCH, I. *Sentence-Transformers: State-of-the-Art Sentence Embeddings using BERT*. 2020. https://github.com/UKPLab/sentence-transformers. Accessed: 2025-07-29.

[Richards 2022]RICHARDS, M. *Software Architecture Patterns: Understanding Common Architectural Styles and When to Use Them*. 2nd. ed. Sebastopol, CA: O'Reilly Media, Inc., 2022. ISBN 978-1-098-13427-3. Disponível em: <https://www.oreilly.com/library/view/software-architecture-patterns/9781098134280/>.

[Ristoski e Paulheim 2018]RISTOSKI, P.; PAULHEIM, H. Rdf2vec: RDF graph embeddings and their applications. *Semantic Web Journal*, 2018. Preprint available online.

[Robertson e Zaragoza 2009]ROBERTSON, S.; ZARAGOZA, H. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, Now Publishers Inc., Hanover, MA, USA, v. 3, n. 4, p. 333–389, abr. 2009. ISSN 1554-0669. Disponível em: <https://doi.org/10.1561/1500000019>.

[Robertson 1977]ROBERTSON, S. E. The probability ranking principle in ir. *Journal of documentation*, Emerald Group Publishing Limited, v. 33, n. 4, p. 294–304, 1977.

[Rumelhart, Hinton e Williams 1986]RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

[Russell e Norvig 2020]RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Pearson, 2020.

[Salton e Buckley 1988]SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, v. 24, n. 5, p. 513–523, 1988.

[Scarselli et al. 2008]SCARSELLI, F. et al. The graph neural network model. *IEEE transactions on neural networks*, IEEE, v. 20, n. 1, p. 61–80, 2008.

[Scarselli et al. 2005]SCARSELLI, F. et al. Graph neural networks for ranking web pages. In: IEEE. *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*. [S.l.], 2005. p. 666–672.

[Shannon 1948]SHANNON, C. E. A mathematical theory of communication. *The Bell system technical journal*, Nokia Bell Labs, v. 27, n. 3, p. 379–423, 1948.

[Shen et al. 2025]SHEN, S. et al. Quantization and retrieval in long-context llms. *arXiv preprint arXiv:2505.20276*, 2025. Disponível em: <https://arxiv.org/abs/2505.20276>.

[Song et al. 2020]SONG, K. et al. MPNet: Masked and permuted pre-training for language understanding. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [s.n.], 2020. Disponível em: <https://arxiv.org/abs/2004.09297>.

[Sun et al. 2024]SUN, J. et al. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In: *Proceedings of the 2024 International Conference on Management of Data (SIGMOD '24)*. [s.n.], 2024. p. 1–26. Disponível em: <https://arxiv.org/abs/2307.07697>.

[Tanon 2025]TANON, T. P. *Oxigraph: A graph database implementing the SPARQL standard*. 2025. Software. Oxigraph version 0.4.11. Disponível em: <https://github.com/oxigraph/oxigraph>.

[The BigScience Workshop 2023]The BigScience Workshop. BLOOM: a 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2023. Disponível em: <https://arxiv.org/abs/2211.05100>.

[Touvron et al. 2023]TOUVRON, H. et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[Vaswani et al. 2017]VASWANI, A. et al. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

[Veličković et al. 2018]VELIČKOVIĆ, P. et al. Graph attention networks. In: *International Conference on Learning Representations (ICLR)*. [s.n.], 2018. Disponível em: <https://openreview.net/forum?id=rJXMpikCZ>.

[Vrandečić e Krötzsch 2014]VRANDEČIĆ, D.; KRÖTZSCH, M. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, ACM, v. 57, n. 10, p. 78–85, 2014.

[Wang et al. 2017]WANG, Q. et al. 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.

[Wang et al. 2020]WANG, W. et al. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *CoRR*, abs/2002.10957, 2020. Disponível em: <https://arxiv.org/abs/2002.10957>.

[Wang et al. 2023]WANG, X. et al. Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases. *arXiv preprint arXiv:2308.11761*, 2023.

[Wang et al. 2024]WANG, Y. et al. Knowledge graph prompting for multi-document question answering. In: *Proceedings of the AAAI conference on artificial intelligence*. [S.l.: s.n.], 2024. v. 38, n. 17, p. 19206–19214.

[Wang et al. 2014]WANG, Z. et al. Knowledge graph embedding by translating on hyperplanes. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2014. v. 28, n. 1.

[Wei et al. 2022]WEI, J. et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[Wolf et al. 2020]WOLF, T. et al. Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, 2020. p. 38–45. Disponível em: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.

[Wu et al. 2020]WU, Z. et al. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, IEEE, v. 32, n. 1, p. 4–24, 2020.

[Yasunaga et al. 2022]YASUNAGA, M. et al. *QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering*. 2022. Disponível em: <https://arxiv.org/abs/2104.06378>.

[Yasunaga et al. 2022]YASUNAGA, M. et al. *QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering*. 2022. Disponível em: <https://arxiv.org/abs/2104.06378>.

[Yih et al. 2016]YIH, W.-t. et al. The value of semantic parse labeling for knowledge base question answering. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. [S.l.: s.n.], 2016. p. 201–206.

[Zhang et al. 2022]ZHANG, X. et al. *GreaseLM: Graph REASoning Enhanced Language Models for Question Answering*. 2022. Disponível em: <https://arxiv.org/abs/2201.08860>.

[Zhang et al. 2018]ZHANG, Y. et al. Variational reasoning for question answering with knowledge graph. In: *AAAI*. [S.l.: s.n.], 2018.

[Zhou et al. 2020]ZHOU, J. et al. Graph neural networks: A review of methods and applications. *AI Open*, Elsevier, v. 1, p. 57–81, 2020.