

Uso de IA Generativa no Apoio à Aprendizagem de Programação

Rafael Almeida de Bem*, Prof^a. Dr^a. Andréa Aparecida Konzen**

*Aluno de Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

**Orientadora, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

Trabalho de Conclusão de Curso

Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul

Resumo—Avanços recentes em inteligência artificial (IA) têm transformado sua aplicação de uma exploração teórica para implementação prática, criando novas oportunidades e desafios em ambientes de aprendizagem. Este trabalho de conclusão de curso, realizado na Pontifícia Universidade Católica do Rio Grande do Sul, explora como a IA generativa pode aprimorar experiências de aprendizagem na disciplina de Fundamentos da Programação. O sistema desenvolvido utiliza grandes modelos de linguagem e emprega conceitos-chave como *transformers*, *zero-shot prompting* e *retrieval-augmented generation* para fornecer assistência personalizada aos estudantes, gerando respostas relevantes e sensíveis ao contexto. Os resultados destacam tanto os benefícios quanto as limitações da IA generativa em âmbitos de aprendizagem.

Palavras-chave—Inteligência artificial generativa, aprendizado em ciência da computação, grandes modelos de linguagem, *transformers*, *retrieval-augmented generation*, respostas sensíveis ao contexto.

Abstract—Recent advances in artificial intelligence (AI) have transformed its application from theoretical exploration to practical implementation, creating new opportunities and challenges in learning environments. This undergraduate thesis, conducted at the Pontifícia Universidade Católica do Rio Grande do Sul, explores how generative AI can enhance learning experiences in the Fundamentals of Programming course. The developed system utilizes large language models and employs key concepts such as *transformers*, *zero-shot prompting*, and *retrieval-augmented generation* to provide personalized assistance to students, generating relevant and context-aware responses. The results highlight both the benefits and limitations of generative AI in improving learning outcomes.

Key-words—Generative artificial intelligence, learning in computer science, large language models, *transformers*, *retrieval-augmented generation*, context-sensitive responses.

I. INTRODUÇÃO

A tecnologia tornou-se parte de todos os aspectos da vida moderna, influenciando indústrias e profissões em todos os setores [1]. No mercado de trabalho, a proficiência em tecnologia não é vista como uma vantagem, mas como uma necessidade, até mesmo em funções que não são diretamente ligadas à tecnologia [2]. Essa necessidade

de habilidades tecnológicas onipresente traz um desafio significativo: aprender a programar.

Programação é uma habilidade presente em grande parte da tecnologia moderna, e aprender a programar não é trivial. A curva de aprendizado íngreme pode ser desanimadora, levando a altas taxas de evasão entre estudantes da computação [3], [4]. Esse desafio se agrava pelo fato de que métodos de aprendizagem tradicionais podem não atender efetivamente às necessidades de aprendizado dos discentes [5]–[9].

Um conceito chave para abordar esse problema é o pensamento computacional, entendido como “uma abordagem à resolução de problemas, *design* de sistemas e compreensão de comportamento humano baseada nos conceitos fundamentais da computação”¹ [10]. Ele fornece uma estrutura que pode ajudar a tornar o aprendizado de programação mais acessível e intuitivo [7], [10], [11], [11], [12].

A inteligência artificial (IA), em específico a IA generativa, se apresenta como uma ferramenta de apoio promissora no aprendizado da programação. A IA se mostra capaz de fornecer assistência personalizada, gerar exemplos e dar *feedbacks* em tempo real, tornando o processo de aprendizagem mais envolvente e menos assustador [7], [13]–[17].

Adicionar uma ferramenta de apoio ao aprendizado personalizada não só melhora os resultados do aprendizado, mas também contribui para um corpo discente mais satisfeito e motivado [8], [9], [18].

Sendo assim, este estudo tem como objetivo investigar a utilização de inteligência artificial generativa como uma ferramenta auxiliar na aprendizagem de programação, examinando os potenciais benefícios e limitações dessa abordagem. É apresentado um sistema que integra materiais didáticos de disciplinas à IA, fomentando ambientes personalizados de aprendizagem e coletando informações valiosas para análise do aprendizado dos discentes por meio do corpo docente. Esse trabalho foi desenvolvido como trabalho de conclusão de curso do curso de Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul.

¹[Computational thinking is] an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing.

Este trabalho está estruturado da seguinte forma: a Seção II. apresenta conceitos fundamentais da inteligência artificial, a Seção III. aborda uma revisão da literatura relacionada ao aprendizado de programação e ao uso de IA generativa na educação e aprendizagem, a Seção IV. consiste no desenvolvimento do trabalho e do sistema apresentado, as Seções V. e VI. consistem em demonstrar como o sistema pode ser implantado, a Seção VIII. traz os resultados obtidos da utilização desse projeto, a Seção IX. engloba as limitações desse estudo, e a Seção XI. expõe ideias para trabalhos futuros.

II. CONCEITOS DE INTELIGÊNCIA ARTIFICIAL

Essa seção apresenta uma breve fundamentação teórica, abordando desde conceitos básicos como inteligência artificial e redes neurais, até conceitos mais avançados como aprendizado de máquina, *transformers* e técnicas de refinamento.

2.1 Inteligência artificial

Inteligência artificial (IA) é definida como “o estudo de agentes que recebem informações do ambiente e realizam ações”² [19]. Além de compreender a inteligência, a IA busca também criar agentes ou entidades inteligentes [19]. Esta é uma área multidisciplinar que inclui subáreas como aprendizado de máquina, redes neurais, processamento de linguagem natural, visão computacional, e robótica. Embora os termos IA e aprendizado de máquina sejam frequentemente utilizados como sinônimos [20], a IA é um campo mais amplo que engloba várias técnicas e métodos. O aprendizado de máquina, por sua vez, é uma subárea da IA que foca em construir sistemas que aprendem a partir de dados. Exemplos de técnicas de IA que não envolvem aprendizado de máquina incluem sistemas especialistas e algoritmos de busca.

2.2 Redes neurais

Redes neurais são sistemas computacionais não-lineares que modelam a maneira como o cérebro humano realiza determinadas tarefas. Elas são compostas por unidades de processamento simples chamadas neurônios artificiais, organizados em camadas e interligados por sinapses ou arestas.

Segundo [21], “uma rede neural é um processador massivamente paralelo composto por unidades de processamento simples que é naturalmente propensa a guardar sabedoria adquirida por experiência e torná-la disponível para uso”³. As unidades de processamento são chamadas de neurônios artificiais e são organizadas em camadas com pelo menos um neurônio cada. A primeira camada é chamada de camada de entrada, a última camada é chamada de camada de saída, e as intermediárias são

chamadas de ocultas. Caso haja mais de uma camada oculta, a rede é chamada de rede neural profunda, e caso haja apenas um neurônio em toda a rede, ela é chamada de *perceptron* [22]. Os neurônios são conectados entre si por arestas (ou sinapses), que possuem pesos que são ajustados durante o treinamento da rede. Existem diversas arquiteturas de redes neurais. A arquitetura *transformer* é usada pelos modelos de linguagem utilizados, e é abordada na Seção 2.3.

O treinamento de uma rede neural consiste em ajustar os pesos dessas conexões entre neurônios de forma que a rede consiga se aproximar de uma função alvo $f(x)$. Na prática, as redes apenas aproximam a função alvo, resultando na função $\hat{f}(x)$. Redes neurais com ao menos uma camada também são conhecidas como aproximadores universais, pois são capazes de aproximar qualquer função contínua $f(x)$ em um intervalo finito [23]. Para que uma rede neural possa aproximar uma função não-linear (contínua), é necessário que a função de ativação dos neurônios seja não-linear.

2.2.1 Função de ativação: A função de ativação define como será calculada a saída de um neurônio dado suas entradas (x_1, \dots, x_n) e seus pesos (w_{k1}, \dots, w_{kn}) . Existem diversas funções de ativação, como a função binária, linear, logística, ReLU, Leaky ReLU, softmax, Swish, GLU, e SwiGLU. Cada função possui características e aplicações específicas, introduzindo não-linearidade na rede e permitindo que ela aproxime funções complexas. As funções relevantes para este trabalho são apresentadas a seguir.

Função Swish: Definida pela Equação 1 e proposta por [24]. A Swish é uma função de ativação suave que tende a ser igual ou melhor do que a ReLU nas áreas testadas [24].

$$\varphi(v) = \text{Swish}(v) = v \cdot \sigma(\beta v) \quad (1)$$

onde $\sigma(v)$ é a função sigmoide logística e β é um hiperparâmetro variável ou constante que controla a inclinação da função. A função Swish é diferenciável em todos os pontos e tem mostrado desempenho superior em várias tarefas de aprendizado de máquina devido à sua propriedade não-monótona [24].

Função GLU (*Gated Linear Unit*): Definida pela Equação 2 e proposta por [25]. A GLU é uma função de ativação que combina a função ReLU com uma porta de ativação sigmoide.

$$\varphi(v) = \text{GLU}(v) = v \cdot \sigma(Wv + b) \quad (2)$$

onde W e b são pesos e *bias* da porta sigmoide, respectivamente, e são parâmetros treináveis.

Função SwiGLU: Definida pela Equação 3, apresentada por [26] e utilizada pelo LLaMA 2 em diante [27], [28]. A SwiGLU é uma função de ativação que combina a função Swish com a GLU, oferecendo o estado-da-arte em desempenho e eficiência.

$$\varphi(v) = \text{SwiGLU}(v) = \text{Swish}_\beta(Wv + b) \otimes \sigma(Vv + c) \quad (3)$$

onde \otimes é o operador de multiplicação matricial elemento a elemento, β , W , b , V e c são hiperparâmetros treináveis.

²[AI is] the study of agents that receive percepts from the environment and perform actions.

³A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use.

Para determinar qual função de ativação utilizar, é necessário considerar a natureza do problema que está sendo resolvido e testar diferentes funções para determinar qual melhor se adapta a função alvo; em outras palavras, qual função de ativação $\varphi(x)$ gera $\hat{f}(x)$ mais próxima de $f(x)$.

2.3 Transformers

Transformers são uma arquitetura de aprendizado profundo proposta por [29] que se destacou em tarefas de processamento de linguagem natural. A arquitetura evita o uso de recorrências, substituindo-as por mecanismos de atenção que permitem a conexão de dependências entre entrada e saída. Além disso, a arquitetura é altamente paralelizável, o que a torna mais eficiente em relação a redes neurais recorrentes (RNN), arquitetura utilizada anteriormente. Redes neurais baseadas em *transformers* são redes totalmente conectadas que seguem uma estrutura de codificação e decodificação similar à utilizada em RNNs, onde o codificador (*encoder*) recebe uma entrada (x_1, \dots, x_n) e a codifica em um vetor $\mathbf{z} = (z_1, \dots, z_n)$, que é então decodificado pelo decodificador (*decoder*) em uma saída (y_1, \dots, y_m) [29]. O conceito de atenção é descrito como um mapeamento de uma consulta e um conjunto de pares chave-valor para uma saída, onde a saída é uma soma ponderada dos valores, e os pesos são determinados por uma função de compatibilidade entre a consulta e a chave correspondente [29]. Segundo [29], *transformers* utilizam a função de ativação *softmax*, uma função de ativação comumente utilizada em redes neurais para produzir distribuições de probabilidade sobre múltiplas classes.

Existem três tipos de *transformers*: *encoder-only*, *decoder-only* e *encoder-decoder*. A arquitetura *encoder-decoder* é aquela proposta por [29], que consiste em camadas de *encoders* que codificam as entradas em *tokens* vetoriais munidos de contexto, e camadas de *decoders* que decodificam esses *tokens* em saídas. O *decoder* é similar ao *encoder* em sua arquitetura com a adição de outra camada de atenção que permite que o *decoder* se concentre em partes específicas da entrada. Para evitar que o *decoder* acesse informações futuras, uma máscara de atenção (*masked self-attention*) é aplicada, onde a atenção é limitada a *tokens* anteriores ao *token* atual [29]. Isso leva a uma arquitetura autoregressiva, onde a saída de um *token* depende apenas dos *tokens* anteriores.

2.3.1 Hugging Face Transformers: Transformers⁴ é uma biblioteca de código aberto desenvolvida pela Hugging Face que implementa modelos de linguagem baseados em *transformers* [29], [30]. A biblioteca é amplamente utilizada em tarefas de processamento de linguagem natural e aprendizado de máquina, e possui uma vasta gama de modelos pré-treinados disponíveis para uso em diversas tarefas. Essa biblioteca facilita tarefas como treinamento, *fine-tuning*, inferência e avaliação de modelos de linguagem, e é amplamente utilizada em pesquisas e aplicações práticas [30].

⁴Disponível em: <https://github.com/huggingface/transformers>

2.4 Aprendizado de máquina

O aprendizado de máquina é uma subárea da IA que estuda como algoritmos podem melhorar seu desempenho em tarefas a partir de experiências passadas [19], [31]. Ele é dividido em três categorias principais: aprendizado supervisionado, não supervisionado e por reforço. O modelo “aprende” após um conjunto de etapas denominado treinamento, que geralmente é dividido em três etapas:

- 1) *Forward*: a entrada é passada pela rede neural, onde os pesos são aplicados e a saída é calculada.
- 2) *Loss*: a saída $\hat{f}(X)$ é comparada com o rótulo real $f(X)$, e a função de erro $J(\hat{f}(X), f(X))$ é calculada.
- 3) *Backward*: o gradiente da função de erro $\nabla J(\hat{f}(X), f(X))$ é calculado e os pesos da rede são ajustados a fim de minimizar a função de erro.

Para que um modelo aprenda, é necessário definir uma função de erro ou custo $J(\hat{f}(X))$, também conhecida como função *loss* ou *cost*, que quantifica o quão bem o modelo está se saindo dada uma saída $\hat{f}(X) = \hat{y}$ e seu vetor de pesos w . Ou seja, a função de erro $J(\hat{y}, f(X))$ é uma medida de quão bem o modelo está se saindo em relação ao rótulo real $f(X)$ e seus pesos w , e o objetivo do treinamento é minimizar essa função para que o modelo se aproxime o máximo possível da função alvo $f(X)$. Em outras palavras:

$$\arg \min_w J(\hat{y}, f(X)) \quad (4)$$

Durante a etapa *backward* no treinamento, o cálculo do gradiente da função de custo é um passo crucial para ajustar os pesos da rede. O gradiente $\nabla J(\hat{f}(X), f(X))$ representa a taxa de variação da função de erro em relação aos pesos w da rede, e é utilizado para saber a direção e a magnitude do ajuste necessário para minimizar a função de erro. Esse processo é realizado utilizando a técnica de retropropagação do erro (*backpropagation*), que aplica a regra da cadeia para propagar os erros da camada de saída até as camadas iniciais, atualizando todos os pesos da rede. Matematicamente, esse processo é descrito pela Equação 5, onde w_t é o vetor de pesos atualizado, w_{t-1} é o vetor de pesos anterior, η é a taxa de aprendizado e $\nabla_w J$ é o gradiente da função de erro [32], [33]:

$$w_t \leftarrow w_{t-1} - \eta \nabla_w J \quad (5)$$

O treinamento de modelos de aprendizado de máquina é um processo iterativo que consiste em passar por todas as etapas mencionadas diversas vezes e com vetores de entrada diferentes até que o modelo atinja um desempenho satisfatório. Esse treinamento é um processo que demanda uma grande quantidade de dados e poder computacional [27], [28], [34]–[36], e é uma das razões que tornam o treinamento de modelos de aprendizado de máquina uma tarefa complexa e custosa.

2.4.1 Aprendizado supervisionado: A máquina é treinada a partir de um conjunto de dados rotulados, onde cada exemplo de entrada é associado a um rótulo ou classe.

2.4.2 Aprendizado não-supervisionado: A máquina é treinada a partir de um conjunto de dados não-rotulados, onde um possível objetivo é encontrar padrões. Nesse caso, a máquina pode vir a fazer suas próprias classificações e predições.

2.4.3 Aprendizado por reforço: O agente aprende a partir de interações com um ambiente, onde ele recebe recompensas ou punições por suas ações, sem receber instruções explícitas de como realizar a tarefa.

2.5 Processamento de linguagem natural

O processamento de linguagem natural (NLP) é uma subárea da inteligência artificial focada na interação entre computadores e linguagem humana. Inclui tarefas como tradução automática, sumarização, análise de sentimentos e geração de textos. Modelos de linguagem (*language model*, LM) e grandes modelos de linguagem (*large language model*, LLM) fazem parte dessa área. De modo geral, LLMs são *transformers decoder-only* que geram saídas coerentes e sensíveis ao contexto [29]. Como LLMs são capazes de gerar textos, eles podem ser considerados modelos generativos.

2.6 Inteligência artificial generativa

Inteligência artificial generativa, ou modelos generativos, são modelos de aprendizado de máquina capazes de gerar novos dados a partir de seus *datasets* de treinamento. Esses modelos são amplamente utilizados em tarefas de geração de texto, imagens e áudio. Exemplos incluem o OpenAI GPT-3 [13] e o Meta LLaMA 3 [28], *transformers* generativos capazes de darem respostas sensíveis ao contexto e em linguagem humana. Para a aprendizagem, esses modelos podem gerar respostas a perguntas de estudantes, gerar materiais didáticos, entre outras aplicações [6], [14], [37].

2.6.1 Generative pre-trained transformers: GPTs são uma classe de LLMs muito utilizados em tarefas de processamento de linguagem natural graças a sua capacidade de gerar textos coerentes e sensíveis ao contexto. Embora “GPT” seja popularmente relacionado aos modelos ChatGPT e às famílias GPT-3 e GPT-4 da OpenAI [13], o termo GPT se refere a uma classe de modelos que utilizam mecanismos de atenção autoregressiva *decoder-only* para gerar textos similares à linguagem humana.

Os GPTs, por serem pré-treinados, possuem um limite máximo de conhecimento: o que foi apresentado durante o treinamento. Isso significa que, caso uma pergunta seja feita a um GPT sobre um assunto que ele não conhece, ele não será capaz de responder corretamente, podendo até halucinar uma resposta [38]. Para que isso não aconteça, são aplicadas técnicas de extensão de conhecimento, como *fine-tuning*, *transfer learning* e *retrieval-augmented generation*.

2.7 Fine-tuning e transfer learning

Fine-tuning e *transfer learning* são técnicas de aprendizado de máquina utilizadas para melhorar o desempenho

de modelos pré-treinados em tarefas e domínios novos. *Transfer learning*, ou aprendizado por transferência, é um conceito amplo que consiste em adaptar um modelo pré-treinado e adaptá-lo a uma tarefa ou domínio novo. *Fine-tuning*, ou ajuste fino, é uma técnica específica de *transfer learning* que consiste em treinar novamente um modelo pré-treinado em um conjunto de dados específico para uma tarefa específica, ajustando seus pesos para a nova tarefa [29], [39]. Essa técnica tem vantagens e desvantagens, e é importante considerar o contexto e a tarefa para decidir se *fine-tuning* é a melhor abordagem. Por exemplo, *fine-tuning* pode ser eficaz quando o conjunto de dados de treinamento é pequeno e a tarefa é similar à original do modelo pré-treinado [39]. Entretanto, *fine-tuning* pode levar a *catastrophic forgetting*, onde o modelo esquece informações importantes do conjunto de dados original [40]. Vale ressaltar que embora *fine-tuning* seja uma técnica eficaz quando se tem um conjunto de dados pequeno, “pequeno” é um termo relativo; quando se trata de LLMs, um conjunto de dados pequeno pode conter bilhões ou até trilhões de *tokens* [27], [28], [39]. Nesses casos, pode ser interessante utilizar *retrieval-augmented generation*.

2.8 Retrieval-augmented generation

Retrieval-augmented generation (RAG) é uma técnica que permite que LLMs referenciem conhecimentos externos ao seu treinamento durante o cálculo de um resultado. Ela é utilizada para melhorar a qualidade e a relevância das respostas geradas por LLMs e diminuir a quantidade de halucinações e respostas incorretas [41]. É importante ressaltar que RAG não é limitada a LLMs e pode ser utilizada em outros âmbitos [41]; entretanto, no contexto desse trabalho, as técnicas de RAG apresentadas serão especificamente relacionadas a LLMs. No contexto de aprendizagem, pode-se utilizá-la para responder perguntas de estudantes utilizando o próprio material didático de determinada disciplina como referência, gerando respostas altamente personalizadas.

A Figura 1 apresenta um exemplo de aplicação do RAG no ChatGPT dada uma *query* (busca, pergunta) que obriga o modelo a buscar informações relevantes em documentos externos pois a informação não está presente em seu conjunto de treinamento. De modo geral, RAG é dividido em três etapas: indexação, recuperação e geração. **Indexação (*indexing*):** documentos relevantes em diversos formatos são “lidos”, processados e transformados em vetores de contexto. Documentos podem ser textos, PDFs, imagens, áudios, e afins. Aqui, “lidos” significa que de alguma forma o conteúdo do documento é convertido em um formato que o sistema de armazenamento vetorial possa entender e processar — por exemplo, um arquivo em PDF é convertido em texto puro e suas imagens são extraídas e transformadas em texto — e então é transformado em um vetor de contexto. Os detalhes de como essa transformação é feita dependem do tipo de documento e do sistema de armazenamento vetorial utilizado e não fazem parte do escopo desse trabalho.

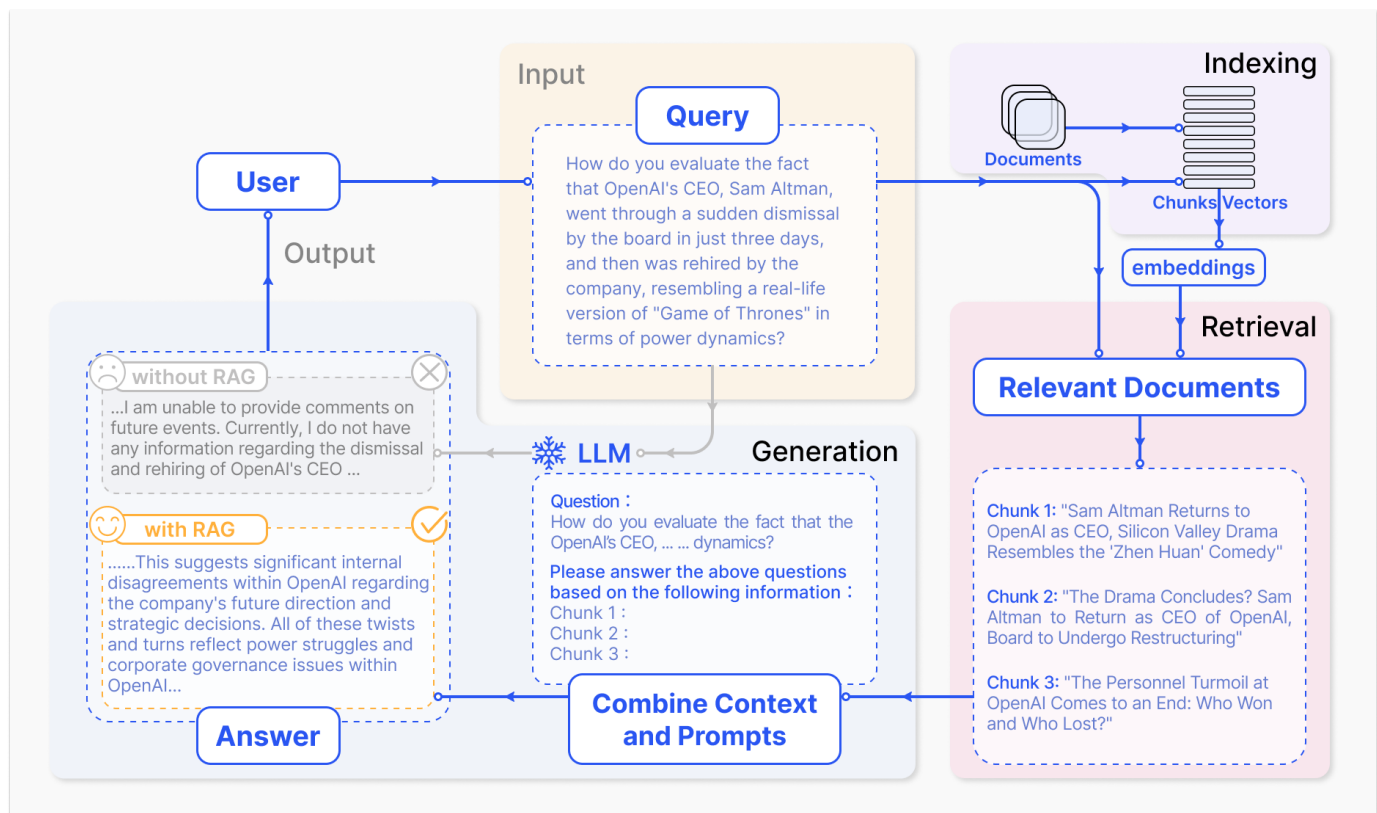


Figura 1. Fluxo de informações com *Retrieval-Augmented Generation* (RAG) e sem RAG em um LLM. O processo de RAG consiste em: (1) indexar documentos relevantes e transformá-los em vetores de contexto; (2) recuperar os k -melhores vetores para uma dada consulta; (3) concatenar os vetores de contexto recuperados com a entrada e repassar ao LLM [42]. Fonte: [42].

Recuperação (*retrieval*): dado um vetor de consulta, o banco de dados vetorial retorna os k -melhores vetores que mais se assemelham ao vetor de consulta. A busca pode ser feita utilizando funções heurísticas, buscas hierárquicas ou outras técnicas. Cabe ao implementador decidir qual a melhor função de busca para um determinado conjunto de documentos e seus vetores.

Geração (*generation*): os vetores de contexto recuperados são concatenados com a entrada e repassados ao LLM para geração de uma resposta enriquecida com o contexto recuperado.

Existem três tipos de RAG: *naïve*, *advanced* (avançado) e modular. *Naïve* RAG é o mais simples e consiste apenas em indexação, recuperação e geração. Esse tipo de RAG pode trazer problemas na qualidade das respostas geradas pois o modelo pode simplesmente repetir o conteúdo recuperado sem entender o contexto e sem adicionar informações. *Advanced* RAG é uma evolução do *naïve* RAG que implementa estratégias de pré-recuperação e pós-recuperação para melhorar a qualidade dos vetores de contexto recuperados, além de aplicar técnicas de janela deslizante para melhorar a indexação. Por fim, RAG modular é uma abordagem mais complexa que divide o processo de RAG em módulos independentes, onde cada módulo é responsável por uma etapa do processo, incorporando diversas estratégias para melhorar o desempenho do sistema como um todo. A Figura 2 apresenta uma

comparação entre RAG *naïve* e avançado, e a Figura 3 apresenta a arquitetura de um RAG modular [41], [42].

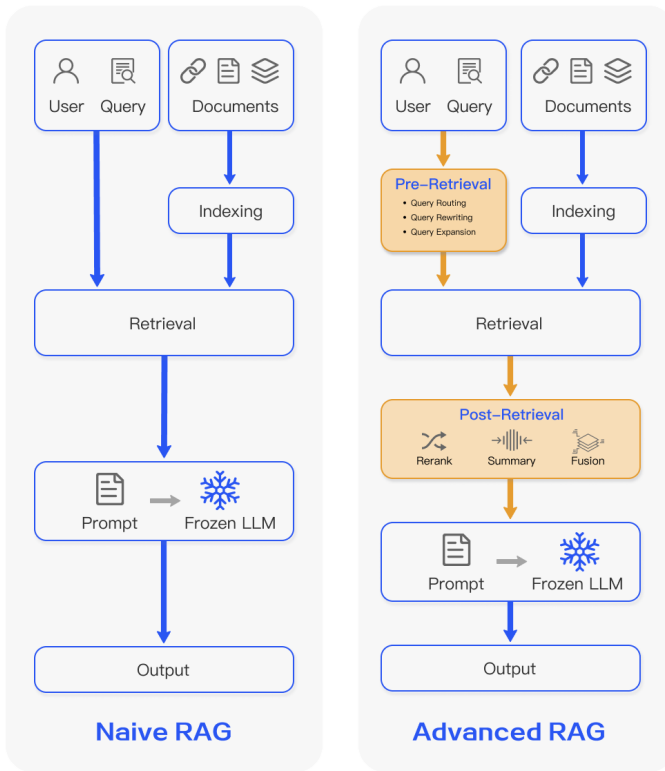


Figura 2. Comparação entre *naive* e *advanced* RAG. É possível observar que o *advanced* RAG possui estratégias de pré-recuperação e pós-recuperação para melhorar a qualidade dos vetores de contexto recuperados. Fonte: [42].

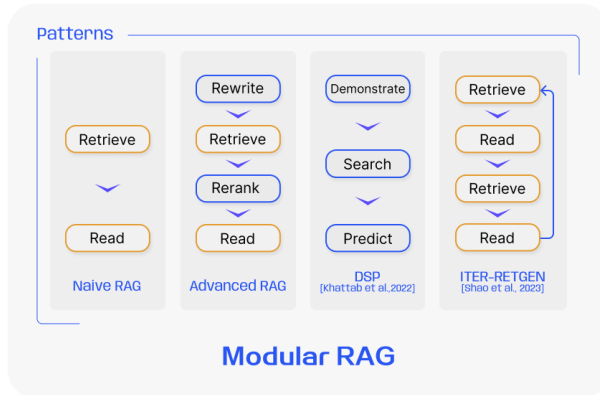


Figura 3. Arquitetura de um RAG modular, onde o processo de RAG é dividido em módulos independentes, cada um responsável por uma etapa do processo. Um RAG modular pode incorporar RAG *naive* e avançado, além de outras estratégias para melhorar o desempenho do sistema como um todo. Fonte: [42].

Outra alternativa para introduzir conhecimento de um domínio previamente desconhecido a um LLM é a utilização de *fine-tuning* e *transfer learning*, onde o modelo é treinado em um conjunto de dados específico para uma tarefa específica [29], [39]. Entre RAG e *fine-tuning*, RAG é uma técnica mais flexível e que não requer um conjunto de dados de treinamento específico, pois o conhecimento é extraído de documentos externos de maneira dinâmica [41], enquanto *fine-tuning* requer um conjunto de dados de treinamento específico para a tarefa desejada e cria um

novo modelo, o que pode ser problemático em um uso onde os documentos relevantes estão sempre mudando.

2.9 Prompt-based tuning em LLMs

Um estudo recente apresenta um novo paradigma para especialização de LLMs modernos, *prompt-based tuning*, ao invés de *fine-tuning* [43]. Em seu estudo, os autores apresentam esse paradigma e o comparam com o anterior, o *pre-train*, *fine-tune*, que consiste em treinar um LLM em um grande conjunto de dados e então ajustar os pesos do modelo para uma tarefa específica. Esse novo paradigma consiste em escrever *prompts* detalhados para resolver uma tarefa específica sem a necessidade de adaptar um LLM para isso. Por exemplo, é possível fazer análise de sentimentos utilizando o modelo com a seguinte *prompt*: “Complete the phrase: ‘I missed the bus today, I felt so _’”, onde o modelo completaria a frase com uma palavra que expressa um sentimento, como “disappointed” [43]. Os estudos [44], [45] apresentam abordagens similares, com resultados alinhados ao apresentado em [43].

Dito isso, *prompt-based tuning* introduz a necessidade de realizar uma engenharia de *prompts*, também conhecido como *prompt engineering*, para obter um resultado efetivo ao questionar o modelo.

III. TRABALHOS RELACIONADOS E REVISÃO DA LITERATURA

Nos últimos anos, houve um avanço significativo no desenvolvimento e aplicação de LLMs, que evoluíram desde modelos de linguagem estatísticos até redes neurais, culminando na era atual de GPTs muito poderosos. Revisões abrangentes feitas em [46] e [47] fornecem uma análise aprofundada desses desenvolvimentos, destacando a evolução desde métodos estatísticos até redes neurais e os avanços subsequentes com GPTs e LLMs. Nelas são abordados aspectos do pré-treinamento, adaptação, utilização e avaliação de capacidade, o que oferece uma visão detalhada do estado-da-arte na pesquisa de LLMs. Os autores apontam que LLMs como BERT, GPT-4, LLaMA, ChatGPT e Claude não apenas melhoraram o desempenho de várias tarefas de PLN, mas também abriram caminho para novas aplicações de IA, transformando significativamente o campo.

A seguir é apresentada uma revisão dos trabalhos relacionados, que abordam o uso de modelos de linguagem na aprendizagem e aprendizagem e o desenvolvimento de metodologias para a aprendizagem de programação.

3.1 Large language models in education: A focus on the complementary relationship between human teachers and ChatGPT (2023)

O trabalho de [14] explora o uso de modelos de linguagem na aprendizagem, com foco na relação complementar entre professores humanos e ChatGPT na aprendizagem de inglês para estrangeiros na Coreia do Sul. Nele, os autores examinam como o ChatGPT pode complementar os papéis dos professores humanos, onde identificaram que

o ChatGPT assume quatro papéis principais: interlocutor, gerador de conteúdo, assistente de aprendizagem e avaliador. Além disso, os autores discutem o papel dos professores em conscientizar os estudantes sobre o uso de ferramentas como o ChatGPT e como eles podem ser usados para melhorar o aprendizado. O trabalho destaca a importância de professores e modelos de linguagem trabalharem juntos para melhorar o aprendizado dos estudantes, e confirma que professores necessitam de treinamento para utilizar ferramentas de IA em sala de aula [14], [15].

3.2 ChatGPT for good? On opportunities and challenges of large language models for education (2023)

O trabalho de [6] investiga as oportunidades e desafios do uso de LLMs, como ChatGPT, na aprendizagem. Os autores discutem como esses modelos podem ser utilizados em diferentes contextos educacionais e momentos de desenvolvimento (infância, adolescência, idade adulta), e como eles podem ser utilizados para melhorar o engajamento dos discentes e personalizar suas experiências. Além disso, discutem a necessidade de desenvolver competências entre professores e alunos para entender e utilizar essas tecnologias de maneira eficaz, ressaltando a importância do pensamento crítico e estratégias de verificação de fatos. O estudo também aborda questões éticas, como o viés potencial nas respostas geradas e a necessidade de supervisão humana contínua, e oferece recomendações para uma integração responsável e ética desses modelos no ambiente educacional [6].

3.3 ChatGPT no auxílio da aprendizagem de programação: um estudo de caso (2023)

O estudo de [37] investiga o uso do ChatGPT como ferramenta auxiliar na aprendizagem de programação. Utilizando uma metodologia quali-quantitativa, os autores analisaram como o ChatGPT pode apoiar os alunos na compreensão de conceitos, fornecendo exemplos de código e feedback personalizado. Os resultados indicam que, embora o ChatGPT seja uma tecnologia promissora para a aprendizagem da programação, ele apresenta limitações, como a precisão das respostas e a necessidade de orientação contínua por parte dos professores para evitar mal-entendidos. O estudo destaca a importância de integrar tecnologias de IA de forma cuidadosa e monitorada para melhorar o aprendizado e o engajamento dos alunos [37].

3.4 Ensino de Programação Orientada a Objetos Para Iniciantes: Uma Metodologia para Programação Criativa (2023)

O artigo de [5] explora metodologias para a aprendizagem de programação orientada a objetos (POO) a iniciantes, especialmente no contexto de ensino remoto durante a pandemia. A metodologia aplicada incluiu a utilização de salas de aula invertidas, metodologias ativas, e integração de projetos práticos, com o objetivo de desenvolver o pensamento computacional dos alunos. Os resultados

destacaram tanto as oportunidades quanto os desafios enfrentados, como restrições tecnológicas e a necessidade de supervisão constante. Segundo [5], os estudantes valorizaram a experiência remota, destacando a qualidade dos conteúdos e dos trabalhos práticos. O estudo enfatiza a importância de futuras pesquisas sobre estratégias de aprendizagem de programação, considerando a proposta metodológica de Programação Criativa como uma abordagem eficaz para promover a aprendizagem integral e o desenvolvimento de habilidades essenciais na área de programação. Também é enfatizado por [5] a importância de desenvolver o Pensamento Computacional (PC) [10] nos alunos, e como a metodologia de Programação Criativa pode ser uma abordagem eficaz para desenvolver essas habilidades.

3.5 Can artificial intelligence transform higher education? (2020)

O artigo de [48] explora o impacto potencial e real da IA no ensino superior. Os autores examinam diversas aplicações de IA, como sistemas tutores inteligentes, análise de aprendizado e personalização adaptativa. Apesar do potencial significativo da IA para transformar práticas de aprendizagem, os resultados empíricos atuais mostram que a influência da IA na melhoria de resultados de aprendizagem ainda é marginal. Os autores destacam a necessidade de maior envolvimento dos educadores no desenvolvimento de IA educacional para garantir que as aplicações futuras abordem habilidades críticas e pensamento criativo [48]. Vale ressaltar que o trabalho de [48] foi publicado em 2020, e desde então novos avanços e pesquisas em IA educacional, além de novos modelos mais poderosos e coerentes, têm sido realizados, o que pode impactar as conclusões apresentadas no artigo.

3.6 Análise dos trabalhos estudados

É possível observar que a utilização de modelos de linguagem na aprendizagem é uma tendência ascendente, com diversos estudos explorando como esses modelos podem ser utilizados para melhorar o aprendizado dos estudantes. Os trabalhos de [6], [14], [37] destacam a importância de professores e IA trabalharem juntos para melhorar o aprendizado dos estudantes, e ressaltam a necessidade de treinamento e supervisão contínua para garantir o uso ético e responsável dessas tecnologias. Já o trabalho de [5] explora metodologias para a aprendizagem de programação orientada a objetos a iniciantes, destacando a importância do pensamento computacional dos alunos e como a aprendizagem de programação não é apenas sobre a linguagem de programação, mas sobre o desenvolvimento de outras habilidades essenciais na área de computação. Por fim, o trabalho de [48] examina o impacto potencial e real da IA no ensino superior, destacando o potencial significativo da IA para transformar práticas de aprendizagem. Esses trabalhos contribuem para a compreensão de como a IA pode ser utilizada para melhorar o aprendizado dos estudantes e como professores e IA podem trabalhar juntos

para alcançar esse objetivo. Em vista disso, este trabalho busca contribuir para essa discussão, explorando como um sistema de IA pode ser utilizado para auxiliar discentes na aprendizagem de programação, fornecendo respostas personalizadas e orientações para suas dúvidas, além de agregar valor à instituição que o adotar.

3.7 Conexões com o trabalho desenvolvido

A integração da IA no aprendizado e processos de aprendizagem tem sido amplamente estudada, com diversas abordagens demonstrando os potenciais benefícios e desafios do uso da IA para melhorar as experiências de aprendizado. Por exemplo, [14] explora a relação complementar entre professores humanos e o ChatGPT na aprendizagem de inglês, enfatizando a necessidade de os professores incorporarem efetivamente ferramentas de IA em suas estratégias de aprendizagem [14]. Este estudo fornece uma base para entender como a IA pode apoiar os educadores, um conceito examinado por este trabalho ao aplicar IA generativa na aprendizagem de programação.

De modo similar, [6] discute as oportunidades e desafios do uso de grandes modelos de linguagem, como o ChatGPT, em vários contextos do aprendizado, destacando a importância das competências de professores e alunos na utilização dessas tecnologias [6]. Este estudo aborda um assunto similar, focado especificamente no domínio da aprendizagem de programação e propondo implementações práticas para apoiar tanto os alunos quanto as instituições de ensino durante sua trajetória acadêmica.

A pesquisa de [37] sobre o uso do ChatGPT para auxiliar a aprendizagem de programação destacou tanto o potencial quanto as limitações desse modelo em fornecer feedback personalizado e exemplos de código [37]. Isso se alinha com o objetivo de utilizar um LLM para oferecer suporte personalizado aos alunos, abordando limitações semelhantes por meio de capacidades avançadas do modelo e implantação estruturada em ambientes educacionais.

As taxas de reprovação em cursos introdutórios de programação foram revisitadas por [11], atribuindo a alta evasão à complexidade dos conceitos e à falta de suporte [11]. Abordando essa lacuna, este estudo visa reduzir as taxas de evasão ao fornecer assistência em tempo real gerada por IA para ajudar os alunos a superar obstáculos de aprendizagem.

No entanto, nenhum desses estudos integra diretamente o material didático do curso, como é feito nesse trabalho. Este estudo se destaca por utilizar a IA generativa para fornecer respostas e orientações baseadas no material específico das disciplinas, criando um ambiente de aprendizagem altamente personalizado e contextualizado.

Ao se basear nesses estudos fundamentais, essa pesquisa visa contribuir para a área de IA na aprendizagem, focando em sua aplicação em cursos de programação. O sistema implementado e sua integração nos *frameworks* educacionais buscam não apenas validar a eficácia da IA em melhorar os resultados de aprendizagem, mas também oferecer uma solução escalável para os desafios generalizados na aprendizagem da ciência da computação.

IV. TRABALHO DESENVOLVIDO

Neste trabalho é apresentado um sistema de apoio ao aprendizado de programação utilizando modelos de linguagem de grande escala capazes de responder perguntas dos estudantes e fornecerem orientações personalizadas baseando-se em materiais didáticos de determinada cadeira. O sistema desenvolvido foi arquitetado para ser modificável e expansível, permitindo a adição de novas funcionalidades, outros modelos, outras interfaces, outros sistemas de RAG, e até mesmo a integração com outras ferramentas de aprendizagem. O sistema possui os seguintes requisitos básicos:

- Responder perguntas utilizando informações externas à seu treinamento através de extensão de conhecimento.
- Evitar dar respostas diretas, tentando aplicar técnicas de aprendizagem guiada.
- Dar respostas coesas e corretas seguindo o material didático.
- Não diminuir a capacidade de inferência do modelo-base.
- Ser de fácil uso e acesso; acesso se dá, por exemplo, por uma interface web.
- Agregar valor tanto ao aprendizado do discente quanto ao trabalho do docente.

As seções seguintes contextualizam o problema abordado e detalham a implementação do sistema.

4.1 Pensamento computacional e evasão em cursos de computação

A evasão nos cursos de computação é um problema recorrente e preocupante, tanto no Brasil quanto no exterior [3], [4], [8], [9], [11]. Diversos fatores contribuem para esse fenômeno, entre eles a dificuldade de aprendizagem dos conceitos fundamentais de programação e lógica [5], [9]. Essas dificuldades são amplamente atribuídas à falta de familiaridade dos alunos com o pensamento computacional [10], uma habilidade essencial que vai além do simples domínio de uma linguagem de programação [12], [17], [18], [49].

O pensamento computacional envolve a capacidade de resolver problemas de maneira lógica e estruturada, habilidades cruciais para o sucesso em qualquer área da computação e outras áreas do conhecimento [10]. No entanto, muitos alunos encontram barreiras significativas ao tentar desenvolver essas habilidades, o que pode levar à frustração e, eventualmente, à desistência do curso ou até mesmo da área. Os estudos de [3], [8], [11] apontam que a evasão em cursos de computação é um problema complexo, influenciado por diversos fatores, sendo apenas um deles a dificuldade de aprendizagem desses conceitos. Dito isso, uma ferramenta que auxilie o professor e o discente na aprendizagem desses conceitos é de grande valia.

4.2 IA no apoio ao aprendizado

A aplicação da IA no apoio ao aprendizado não é uma novidade, mas suas possibilidades estão se expandindo

rapidamente [48]. Embora uma revisão sistemática de pesquisas sobre aplicações de IA no ensino superior sugira que a integração efetiva de IA pode melhorar os resultados de aprendizagem ao fornecer suporte personalizado e adaptativo aos estudantes [50], [48] apresenta um cenário mais complexo, onde seus resultados mostram que a influência da IA obtém melhorias marginais. Reforça-se que o papel dos educadores é crucial para a implementação bem-sucedida dessas tecnologias [6], [14], [17].

O pensamento computacional é fundamental para o sucesso dos alunos não só em cursos de computação mas em diversas outras áreas que possivelmente nem existam ainda [1], [10]. Diversas estratégias podem ser empregadas para ensinar essas habilidades em diferentes contextos e ambientes de aprendizagem. Se cita:

Integração de conceitos computacionais no currículo: Estudos mostram que incorporar conceitos computacionais no currículo do ensino fundamental e médio pode ajudar a desenvolver habilidades de pensamento computacional em estudantes de todas as idades [12], [17], [18], [49].

Aprendizagem baseada em projetos: Projetos práticos que envolvem a criação de programas ou a resolução de problemas complexos incentivam os alunos a aplicar o pensamento computacional de forma criativa e prática. Este método não apenas reforça os conceitos aprendidos, mas também promove habilidades de colaboração e comunicação [1], [5], [17], [18].

Feedback e avaliação contínua: O uso de sistemas de avaliação contínua e feedback imediato pode ajudar os alunos a identificar suas dificuldades e melhorar suas habilidades de pensamento computacional ao longo do curso. A integração com sistemas de IA pode fornecer suporte personalizado e orientações específicas para cada aluno dadas suas necessidades e dificuldades [6], [14], [37]. Nesse sentido também é possível citar a importância de ferramentas de apoio à aprendizagem, como o sistema desenvolvido neste trabalho, que podem fornecer dados valiosos para análise de aprendizado (*learning analytics*) e ajudar a identificar padrões e tendências no desempenho dos alunos [14], [48].

Treinamento de educadores: A eficácia da aprendizagem de pensamento computacional também depende da preparação dos educadores. Oferecer treinamento adequado para professores é crucial para que eles possam integrar essas estratégias em suas aulas de maneira eficaz [6], [14].

Empregar IA na aprendizagem também se faz atraente pois pode fornecer suporte personalizado e adaptativo aos estudantes, ajudando a superar obstáculos de aprendizagem de maneira individual e sem julgamentos [7], [50].

4.3 Escolha do modelo de IA

Existem muitos LLMs disponíveis, cada um com suas características e limitações [47], [51]. Para manter o estudo alinhado com o mercado e não correr tanto risco de expor os usuários a modelos inseguros, foram considerados

apenas aqueles vindos de empresas de renome (Google, Microsoft, Meta, *etc.*) com ampla documentação e abordagens éticas e de segurança bem definidas. Desses, foram descartados modelos de código-fechado como toda a família de modelos da OpenAI: GPT-3 [13], GPT-4, ChatGPT, entre outros. Entre os modelos de *open source* (código aberto), a família de modelos LLaMA se destacou por sua performance excelente em diversos *benchmarks* (testes padronizados) e se tornou muito popular na indústria e pesquisa [27], [34], [47]. No início dessa pesquisa, o modelo LLaMA 3 ainda não havia sido lançado; mesmo assim, a família LLaMA já havia sido escolhida como a melhor opção para o desenvolvimento do sistema desenvolvido [27], [34], [47]. Com o lançamento do LLaMA 3, a escolha foi confirmada, e essa família foi escolhida [28], [52]. Outros motivos que reforçam nossa escolha dessa família são sua documentação extensa e, de muita importância, a ênfase que a equipe de desenvolvimento dá à ética e segurança em seus modelos [28], [52], [53].

Não se levou em consideração apenas os resultados das *benchmarks* auto-reportados pelas equipes de desenvolvimento de cada modelo. Como já haviam sido descartados modelos de código fechado, a utilização dos resultados de *benchmarks* de terceiros se tornou possível, especialmente os realizados por [47], o Open LLM Leaderboard [54] da HuggingFace e o AlpacaEval 2.0 [55]. Também foram removidas as versões quantizadas, modificadas ou com *fine-tuning* da busca, mas foram mantidos aqueles modelos modificados (*fine-tuned*) para tarefas específicas como *chat* e código.

4.3.1 Open LLM Leaderboard: O Open LLM Leaderboard é uma plataforma que automaticamente avalia modelos abertos utilizando técnicas conhecidas de avaliação como ARC [56], HellaSwag [57], MMLU [58] e WINOGRANDE [59]. Também utilizam métricas para avaliar conhecimentos em áreas específicas como matemática (GSM8K [60]) e falácias (TruthfulQA [61]). Como o Open LLM Leaderboard avalia muitos modelos diferentes, foi utilizado um filtro para remover aqueles que são previamente treinados em tarefas específicas (*fine-tuned on domain-specific datasets*, em inglês) e mantidos apenas modelos pré-treinados para que a comparação fosse justa e os resultados refletissem a capacidade dos modelos de aprender e generalizar conhecimento [54]. No Open LLM Leaderboard, o LLaMA 3 70B se coloca em primeiro lugar na média, com um escore de 73.96. Seguido em perto por Qwen 72B com um escore médio de 73.6 e uma versão modificada do Phi 3 com um escore de 73.57. Outros modelos considerados foram o Deepseek 67B (escore de 69.38) e o Mixtral 8x7B (escore de 68.42).

4.3.2 AlpacaEval 2.0: O AlpacaEval 2.0 é uma plataforma de avaliação de modelos de linguagem automática que utiliza um conjunto de avaliação próprio e aberto que avalia LLMs entre si, comparando suas respostas com as respostas geradas pelo OpenAI GPT-4 Preview; o escore é chamado de *win rate*, ou taxa de vitória [55]. Quando comparado com o Open LLM Leaderboard, o AlpacaEval 2.0 é mais focado em avaliar a capacidade dos modelos entre si,

enquanto o Open LLM Leaderboard é mais abrangente e avalia uma variedade de tarefas [54], [55]. No AlpacaEval 2.0, o LLaMA 3 70B se coloca décimo lugar com uma taxa de vitórias de 34.4%. Os modelos que se colocam acima do LLaMA são o OpenAI GPT-4 e suas variações, 01.AI Yi Large, Anthropic Claude 3 Opus e Qwen 1.5 72B. Como um dos requisitos é utilizar modelos de código aberto, foram descartadas as famílias OpenAI GPT-4 e Anthropic Claude 3. Entre os modelos restantes, o LLaMA 3 70B se destaca por possuir ampla documentação, fundamentação científica e ênfase em segurança e ética [28], [52].

4.4 LLaMA 3

A família de modelos escolhida foi a Meta LLaMA 3, sucessora da LLaMA 2 e Codellama. Dentro da família LLaMA 3 existem dois modelos-base ou fundamentais: `llama3:8b` e `llama3:70b`, que se diferenciam pela quantidade de parâmetros e, conseqüentemente, pela capacidade de processamento e requisitos computacionais [28], [52]. O modelo `llama3:8b` possui oito bilhões de parâmetros e é menor e mais rápido, enquanto o `llama3:70b` é maior, com setenta bilhões de parâmetros, e mais poderoso, mas também mais exigente em termos de recursos computacionais [28]. Existe também o `llama3:400b`, com quatrocentos bilhões de parâmetros e quase seis vezes maior que o `llama3:70b`, que foi apenas anunciado e ainda não está disponível para uso [62]. As variações de número de parâmetros refletem diretamente na capacidade de processamento e na qualidade das respostas geradas pelo modelo, assim como em seus requisitos computacionais. Para o uso em tarefas específicas, como *chat* e código, são utilizadas as versões refinadas desses modelos, que são *fine-tuned* para essas tarefas [28], [52]. A Figura 4 mostra uma comparação reportada pela Meta AI entre o LLaMA 3 70B Instruct e outros LLMs.

Para *chats* e conversas há a variação *instruct*, como o `llama3:70b-instruct`. Diferente do LLaMA 2, o LLaMA 3 não possui uma variação específica para código, mas é possível utilizar o `llama3:70b-instruct` para essa tarefa [27], [28]. Para código, não há um refinamento específico do LLaMA 3. No entanto, existe o Codellama. O Codellama é a versão *fine-tuned* do LLaMA 2 para tarefas de código [63], e foi testado neste trabalho como uma alternativa aos modelos de conversação.

O LLaMA 3 possui conteúdos oficiais disponíveis em [28] e [52] que contém informações importantes sobre o modelo, como sua arquitetura e funcionamento. Similar ao LLaMA 2, o LLaMA 3 utiliza *grouped query attention* (GQA) como seu mecanismo de atenção. No entanto, o LLaMA 3 aplica a GQA em ambos os tamanhos de modelo, enquanto o LLaMA 2 o aplica apenas nos modelos com 34 bilhões de parâmetros ou mais [27], [28]. O *tokenizer* do LLaMA 2 é o mesmo do LLaMA 1, com um vocabulário de 32 mil *tokens* [27]. Já o *tokenizer* do LLaMA 3 possui 128 mil *tokens*, o que leva a uma melhoria substancial na performance de codificação (*encoding*) [28]. Em sua arquitetura baixo nível, o LLaMA utiliza a função de ativação SwiGLU

(Equação 3) e normaliza as saídas da rede antes de utilizar a função *softmax* no resultado.

O *dataset* de treinamento do LLaMA 3 é sete vezes maior que o do LLaMA 2, com mais de 15 trilhões de *tokens* [28] e incluindo quatro vezes mais *tokens* de código. Desses 15 trilhões de *tokens*, em torno de 5% são de línguas diferentes do inglês, o que torna o LLaMA 3 um modelo multilíngue [28]. Ou seja, mesmo que o modelo possa ser considerado multilíngue, não se espera que sua performance em outras línguas seja tão boa quanto em inglês. Mesmo assim, nos testes conduzidos neste estudo, o LLaMA 3 se saiu bem em responder perguntas em português.

LLMs possuem um número máximo de *tokens* que são considerados por vez durante a inferência. Isso se chama tamanho ou janela de contexto, e varia de modelo a modelo. O tamanho dessa janela determina quanto do texto anterior o modelo irá “lembrar”, ou levar em consideração, ao gerar uma resposta coerente [29], [39], [64].

Ambos modelos (8B e 70B) possuem um tamanho máximo de contexto de 8 mil *tokens*, mas é possível aumentar esse tamanho através de técnicas como LoRA e QLoRA, cujos conceitos não fazem parte do escopo deste trabalho. Utilizando QLoRA, [65] reporta que foi possível aumentar o tamanho do contexto do LLaMA 3 8B Instruct para 80 mil *tokens* com pouca perda de acurácia, e afirma que talvez seja possível aumentar o contexto ainda mais.

Segundo [28], o treinamento dos modelos LLaMA 3 8B e 70B durou 1.3 milhão e 6.4 milhões de horas respectivamente em dois *clusters* com 24 mil GPUs NVIDIA H100 com 80 GB de VRAM cada. Cada uma dessas GPUs é precificada em torno de 30 mil dólares americanos, e cada cluster possui 24.576 GPUs, o que resulta em um custo apenas em GPUs, sem contar outros requisitos de infraestrutura, de quase um bilhão e meio de dólares americanos [66]. Ou seja, o treinamento desses modelos é extremamente caro, exige uma infraestrutura computacional de ponta e recai sobre grandes empresas e instituições de pesquisa.

4.4.1 Segurança: Sobre segurança e ética, [53] apresenta o Llama Guard, um LLM “guardião” capaz de detectar e corrigir vieses, discurso de ódio, desinformação, entre outros. Em [28] é sugerido que o LLaMA 3 seja utilizado em conjunto com o Llama Guard para garantir a segurança e ética do sistema que será desenvolvido, onde o Llama Guard atua como um filtro de segurança tanto na entrada quanto na saída do sistema. A segurança de LLMs é um fator importantíssimo a ser considerado, especialmente em ambientes de aprendizagem, onde a proteção dos dados dos estudantes e a garantia de respostas seguras e confiáveis são essenciais [28], [52], [53].

4.5 Execução local de um LLM

Existem diversas plataformas e ferramentas que permitem a execução de LLMs localmente. Para isso, foi escolhido o Ollama⁵, uma plataforma de código aberto que permite

⁵Disponível em: <https://ollama.com>

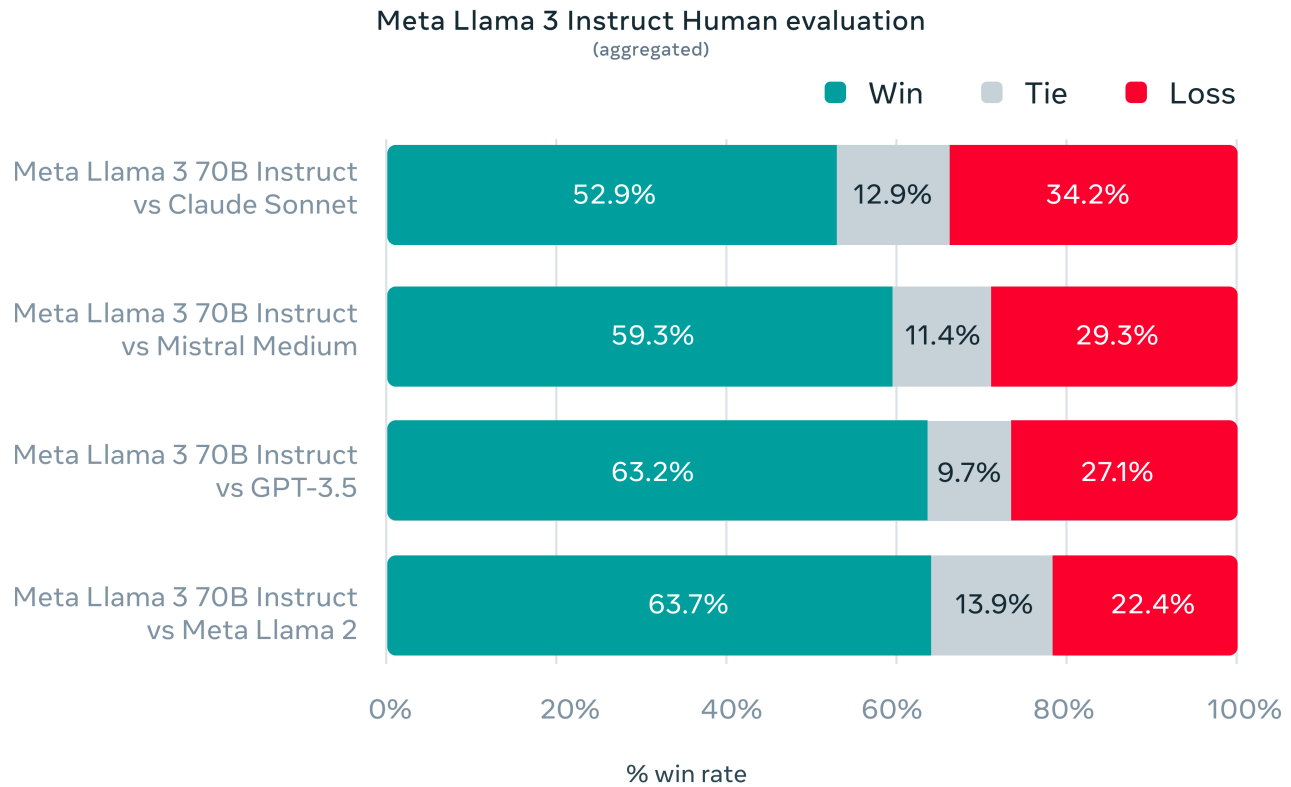


Figura 4. Comparação reportada pela Meta AI entre o LLaMA 3 70B Instruct e outros LLMs. Fonte: [28].

a execução de modelos de IA, principalmente LLMs, em *hardware* local. Outra opção é o vLLM⁶ [67], ferramenta de código aberto que se mostra mais performática que o Ollama, mas com um *setup* um pouco mais envolvente [67]. A vantagem do vLLM é que ele é capaz de paralelizar tarefas de inferência, enquanto o Ollama não⁷. Uma outra ferramenta considerada foi o llama.cpp [69], ferramenta de inferência em C/C++, mas foi descartada pois o Ollama a utiliza como base. O sistema implementado é relativamente indiferente ao executor do modelo, sendo necessárias apenas algumas mudanças na configuração e fora do escopo do estudo.

4.5.1 Nota importante sobre o execução remota e dados sigilosos: Em casos de implantação deste sistema em ambientes de execução remotos, faz-se necessário adotar uma atitude cuidadosa quanto à segurança dos dados que estão sendo transmitidos entre componentes do sistema. Se optar-se por utilizar uma plataforma de hospedagem ou inferência *as a service*, os dados em trânsito ficam expostos — apesar da criptografia existente e outros mecanismos de segurança — ao mundo externo, trazendo riscos à organização. Outro ponto importante é a coleta de dados dos usuários: mesmo se a plataforma escolhida não estiver no Brasil, a Lei Geral de Proteção de Dados pode se aplicar,

abrindo caminho para possíveis ações judiciais [70].

4.6 Métricas de avaliação

A avaliação do sistema se dá pela análise da qualidade das respostas do modelo a perguntas predeterminadas. São utilizadas técnicas de avaliação automática, como a presença de palavras-chave esperadas na resposta, e técnicas de análise humana (*human feedback*). O conjunto de perguntas escolhido é apresentado a seguir:

- 1) O que é um laço?
- 2) Não entendi algoritmos. Poderia me explicar?
- 3) Me dê um exemplo de do-while.
- 4) Meu código não está funcionando.
- 5) O que são métodos? Como os utilizo?
- 6) Qual é a vantagem de usar métodos em programação?
- 7) O que é um erro de compilação?
- 8) Por que meu programa está travando?
- 9) Por que meu programa dá o resultado errado mesmo compilando?
- 10) Para que serve a classe Scanner?
- 11) Como eu imprimo na tela?
- 12) O que são ifs aninhados?
- 13) Como eu escrevo um algoritmo para resolver um problema?
- 14) O que é uma classe?
- 15) Como eu crio e uso classes no meu programa?

⁶Disponível em: <https://github.com/vllm-project/vllm>

⁷Os desenvolvedores do Ollama estão trabalhando para adicionar suporte a paralelismo, mas não foi lançado ainda [68].

- 16) O que é um array?
- 17) O que é uma rede neural?

Utilizando um *script*, o modelo foi questionado cem vezes com cada pergunta e suas respostas foram salvas. Feito isso, as respostas foram avaliadas tanto manualmente quanto automaticamente, por meio de outra instância de LLM.

Também foram utilizadas as ferramentas de avaliação disponibilizadas pelo LlamaIndex. Com elas, foi possível gerar um *dataset* de perguntas a partir dos documentos carregados e avaliar as respostas de maneira automática. As perguntas geradas foram utilizadas para avaliar a relevância dos contextos utilizados e a fidelidade das respostas do modelo a essas perguntas.

Sempre que alguma modificação era feita no sistema, o *script* era executado novamente para avaliar a qualidade das respostas geradas. Isso permitiu aprimorar o sistema de forma iterativa, uma abordagem comum e eficaz para projetos de IA.

4.6.1 Protocolo de teste manual: O protocolo para testes conduzidos manualmente foi o seguinte: a partir da interface web, o modelo era questionado com uma das perguntas acima e a resposta era avaliada em relação à sua coesão, acurácia e relevância. A coesão da resposta se refere à sua estrutura e organização, enquanto a acurácia se refere à correção dos fatos apresentados. A relevância da resposta se refere à sua relação com a pergunta feita.

4.7 Programação do sistema

Toda a codificação do projeto está feita em Python⁸, uma linguagem de programação de alto nível muito utilizada para projetos que envolvam inteligência artificial e aprendizado de máquina. As bibliotecas relevantes utilizadas no projeto são o LlamaIndex e HuggingFace Transformers. Bibliotecas como PyTorch, NumPy e afins são instaladas como dependências dessas bibliotecas principais. Como se possui uma GPU da NVIDIA, a “biblioteca” CUDA foi instalada — assim como o PyTorch com suporte a CUDA — para que fosse possível utilizar a GPU nas tarefas de treinamento e inferência. Localmente, o Mamba (similar ao Conda) foi utilizado para gestão das dependências.

A biblioteca LlamaIndex é utilizada para implementar o RAG.

4.8 Técnicas de extensão de conhecimento

Estender o conhecimento do modelo escolhido com os materiais didáticos disponíveis apresentou um desafio interessante: existem diversas maneiras de se fazer isso, e cada uma delas tem suas vantagens e desvantagens, o que fez esse processo ser extremamente iterativo e experimental. Inicialmente, foi feita uma análise dos materiais didáticos que seriam utilizados para extensão do conhecimento do modelo. Concluída essa análise, foram levantadas as técnicas compatíveis com nosso *dataset* e que trariam bons resultados sem a necessidade de muito poder computacional

ou pré-tratamentos extensos (como rotulação manual, por exemplo). Entre as técnicas no espaço de busca, foi escolhido o RAG como técnica de extensão do conhecimento do modelo. Também foi considerado utilizar *fine-tuning*, onde a documentação do LLaMA 3 indica *parameter-efficient fine-tuning* (PEFT) com LoRA ou QLoRA antes de um processo que modifique todos os parâmetros para não ser necessário um poderil de recursos elevado.

Como mencionado anteriormente, o *fine-tuning* envolve o re-treinamento do modelo que está sendo refinado. Além disso, sempre que forem adicionados novos conteúdos ao domínio — por exemplo, um novo material de aula —, é necessário o re-treinamento. No entanto, o *fine-tuning* entrega um modelo pronto que não requer outras ferramentas para funcionar corretamente. Mesmo assim, o RAG se mostra como a alternativa mais atraente por diversos motivos; entre eles:

- 1) Não é necessário treinar um modelo massivo novamente, poupando recursos financeiros e computacionais.
- 2) Permite a adaptação rápida a novos materiais didáticos.
- 3) É de relativamente simples implementação e manutenção.

Em outras palavras, o *fine-tuning* congela o conhecimento de um modelo naquilo que está codificado na representação numérica de seus pesos. Para adicionar conhecimento, é necessário ou treinar o modelo novamente ou utilizar RAG. A desvantagem do RAG é que se faz necessária uma ferramenta auxiliar que faça essa extensão de conhecimento, adicionando um ponto a mais de falha na infraestrutura.

4.8.1 Sobre fine-tuning: O *fine-tuning* não foi escolhido mas também não foi descartado, pois ele é útil quando somente com RAG e *prompt engineering* o modelo não performa como desejado, seja por questões comportamentais, de conhecimento, segurança, entre outros. A Meta AI disponibiliza um *script* oficial em seu repositório no GitHub para essa finalidade, assim como alguns *datasets* de exemplo [52]. Ao ler a documentação, se cita que “[Utilizar] *fine-tuning* com PEFT LoRA [...] usando o *dataset* OpenAssistant [e um tamanho de *batch* de 4] [...] demora em torno de dezesseis horas em uma única GPU e utiliza menos de 10 GB de memória de GPU; mudar o tamanho de *batch* para 8/16/32 utiliza 11/16/25 10 GB de memória de GPU”. Com PEFT QLoRA demora em torno de seis horas e meia em uma única GPU e 11 GB de memória de GPU. A documentação cita também o projeto Axolotl [71], uma ferramenta que visa facilitar o processo de *fine-tuning* que suporta diversos tipos de *datasets* diferentes. No término do processo de *fine-tuning* ou de *transfer learning* é retornado um modelo com sua arquitetura e pesos ajustados. Esses pesos são o que fazem o modelo; sem eles, não há “modelo”. São esses pesos que são distribuídos, assim como outras informações (exemplo: arquitetura), em plataformas como a HuggingFace. Apesar de não ter sido escolhido nesse momento, aplicar *fine-*

⁸Disponível em: <https://www.python.org>

tuning pode ser uma boa opção para melhorias desse projeto no futuro.

4.9 Sobre RAG

Para implementação do RAG foi utilizada a biblioteca LlamaIndex⁹, uma biblioteca de código aberto e de ampla utilização no mercado [72]. Existem outras bibliotecas de RAG como a LangChain¹⁰ [73], mas o LlamaIndex foi escolhida por ter uma documentação aparentemente mais clara e coesa. Tanto a LangChain quanto o LlamaIndex são bibliotecas conhecidas e utilizadas no mercado, então a escolha entre uma ou outra se deu por preferência pessoal.

Dentro da área de RAG existem diversas estratégias de busca e, similar ao restante do trabalho, escolher qual estratégia utilizar se deu por um processo iterativo. Antes de determinar qual estratégia adotar, foi necessário escolher um modelo de *embedding* que lidasse bem com dados textuais em português, inglês e código.

Brevemente e em altíssimo nível, um *embedding* é uma representação de um texto (nesse contexto) em um vetor numérico. Um modelo de *embedding* é aquele que recebe texto como entrada e retorna um vetor de números que tentam capturar dados semânticos da entrada, permitindo assim uma busca semântica nos dados codificados. Por exemplo, se um usuário fizer uma pergunta sobre laços, o vetor de *embedding* resultante da pergunta será muito próximo do vetor de *embedding* de textos que mencionam laços.

4.10 Escolha do modelo de *embedding*

Foram considerados os modelos de *embedding* mais conhecidos pela comunidade atualmente. São eles:

- nomic-ai/nomic-embed-text-v1.5 [74]
- BAAI/bge-m3 [75]
- mixedbread-ai/mxbai-embed-large-v1 [76]
- intfloat/multilingual-e5-large [77]
- intfloat/e5-mistral-7b-instruct [78]

Em seus testes, [79] compara esses e outros modelos. Seus resultados demonstraram que o BAAI/bge-m3 parece ser a melhor opção. No entanto, [79] não testou *corpus* em português, então podem haver divergências da realidade. Foi escolhido o modelo BAAI/bge-m3 para nosso caso de uso.

4.11 Preparação dos dados

Foram coletados os materiais didáticos da cadeira de Fundamentos da Programação, todos slides em formato PDF, para adicionar ao RAG. Entretanto, alguns dos slides continham texto em formato de imagem, então foi necessário utilizar uma ferramenta de OCR nesses arquivos específicos. No total, são menos de 100 MB de dados crus.

O LlamaIndex disponibiliza diversas formas de carregar documentos para serem adicionados ao banco vetorial.

Entre elas, está sendo utilizada uma API da LlamaCloud¹¹ chamada LlamaParse, dos mesmos mantenedores do LlamaIndex. Se optou pela LlamaParse pois ela possui um plano grátis, limitado a um número de páginas por dia com suporte a OCR e a diversas línguas, e também pode retornar os PDFs em formato Markdown. Poderiam ter sido utilizadas ferramentas locais e de código aberto para atingir resultados semelhantes, mas optou-se por utilizar essa ferramenta, que apresentou excelentes resultados.

O retorno em Markdown pode não parecer muito relevante a primeira vista, mas ele permite que seja feita uma revisão manual desses documentos antes de seguir para a próxima fase do processo, o armazenamento em um banco vetorial. Essa revisão manual se fez necessária; os arquivos retornados foram verificados e modificados sempre que necessário (como erros de ortografia, formatação, ou até mesmo halucinações). Por exemplo, a Figura 5 foi um resultado obtido ao usar a LlamaParse em um dos PDFs, e a Figura 6 é o que se esperava desse mesmo arquivo. Em contrapartida, a Figura 7 apresenta um exemplo de um resultado inalterado vindo diretamente dessa ferramenta.

```

1  # Instruções de Saída
2
3  22
4  #
5
6  GMLmzd2
7
8  6C)- Criação das variáveis
9
10 OLw-#y
11
12 2 mc?
13
14 2Jx

```

Figura 5. Resultado errôneo vindo do LlamaParse.

Feita essa limpeza e revisão, se fez a inclusão desses documentos no banco de dados vetorial do LlamaIndex. Mais uma vez, existem diversas formas de realizar essa importação. Aqui são utilizadas diversas estratégias, ou transformações, de importação de dados: quebra do texto em *chunks* (pedaços) menores, divisão hierárquica dos nodos, e afins. A Figura 8 apresenta um exemplo de importação desses arquivos em Markdown, assim como suas transformações.

Analisando a Figura 8, se vê que está sendo criado um **VectorStoreIndex** da biblioteca LlamaIndex chamado **index**, e é a partir dele que será criado o motor de *chat* ou conversação utilizado pela Pipeline nas próximas seções.

4.12 LlamaIndex e adicionais

A implementação utilizando somente LlamaIndex, por si só, já possui funcionalidades suficientes para atender o

⁹Disponível em: <https://www.llamaindex.ai>

¹⁰Disponível em: <https://www.langchain.com>

¹¹Disponível em: <https://cloud.llamaindex.ai>

```

1  # Instruções de saída
2
3  Como o usuário deve informar a saída (tela)
4
5  System.out.print("Informe o nome:");
6  System.out.print(variavel);
7
8  # Criação das variáveis e constantes
9
10 Constantes são valores que não se alteram:
11
12 - Por convenção, seus identificadores são
13   ↳ escritos em caixa alta.
14 - A declaração de constantes exige a palavra
15   ↳ reservada final.
16
17 final tipo identificadorConstante = valor;
18 Exemplo: final double TAXA = 25;

```

Figura 6. Resultado do LlamaParse esperado do mesmo arquivo utilizado na Figura 5.

```

1  Tipos de Erros
2
3  - Erro de Sintaxe: ocorre quando há um erro
4    ↳ na estrutura do código, como falta de
5    ↳ ponto e vírgula.
6  - Erro de Lógica: ocorre quando o código
7    ↳ está correto, mas a lógica implementada
8    ↳ não produz o resultado esperado.
9  - Erro de Tempo de Execução: ocorre durante
10   ↳ a execução do programa, como divisão por
11   ↳ zero.
12 - Erro de Semântica: ocorre quando o código
13   ↳ está correto, mas não faz o que era
14   ↳ esperado

```

Figura 7. Resultado do LlamaParse sem alterações. O sistema que adicionou o prefixo “Erro de” nos itens da lista nesse caso, pois fora instruído que eram slides de conteúdo de faculdade. Isso pode indicar que o conteúdo dos arquivos está sendo analisado por um LLM e não uma ferramenta de OCR.

requisito básico deste estudo, que é utilizar inteligência artificial como apoio ao aprendizado de programação. Para tal, basta fazer uso da `VectorStoreIndex` apresentada na Figura 8 e convertê-la para um motor de conversação com a função `VectorStoreIndex.as_chat_engine`. A Figura 9 apresenta um exemplo de uso, onde a resposta obtida foi:

Um algoritmo é um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito. Ou, de outra forma, é um texto que contém uma série de ações primitivas que, quando executadas, operam sobre os dados, transformando-os em saídas.

(LLaMA 3, ReAct)

O modelo gerador da resposta anterior foi o LLaMA 3

```

1  from llama_index.core import (
2      VectorStoreIndex,
3      SimpleDirectoryReader
4  )
5  from llama_index.core.node_parser import (
6      HierarchicalNodeParser,
7      MarkdownNodeParser
8  )
9
10 reader = SimpleDirectoryReader('data/md/')
11 index = VectorStoreIndex.from_documents(
12     reader.load_data(),
13     transformations=[
14         MarkdownNodeParser(),
15         HierarchicalNodeParser.from_defaults(
16             chunk_sizes=[2048, 512, 128],
17             chunk_overlap=30
18         ),
19     ],
20 )

```

Figura 8. Leitura dos arquivos em Markdown para serem adicionados ao banco vetorial (`index`) com transformações de quebra hierárquica.

70B Instruct sem quaisquer modificações de hiperparâmetros ou *prompt engineering*, munido com o contexto dos conteúdos de aula mencionados na Seção 4.11.

Importante mencionar a opção `chat_mode` do motor de conversação. A documentação do LlamaIndex entra em detalhes sobre cada opção. A opção `ChatMode.REACT` usa o motor de busca como ferramenta de apoio ao LLM, decidindo se vai ou não fazer uma busca contextual. Essa opção permite que o modelo responda a perguntas que não estão explicitamente definidas em seus provedores de contexto. A opção `ChatMode.CONDENSE_PLUS_CONTEXT` condensa o histórico da conversa com o usuário em uma única pergunta, busca em seus provedores de contexto informações adicionais sobre aquele assunto e então repassa ao LLM o contexto encontrado e a mensagem *verbatim* do usuário. Ambas opções foram testadas e ambas trouxeram bons resultados. A Figura 10 apresenta um exemplo de uso da opção `ChatMode.CONDENSE_PLUS_CONTEXT`, onde a resposta obtida foi:

Um algoritmo é um conjunto finito de regras bem definidas para a solução de um problema em um tempo finito. Além disso, também é definido como um texto (como uma receita de bolo) onde cada linha contém uma ação primitiva (ação elementar passível de execução por um humano ou uma máquina).

(LLaMA 3, Condense Plus Context)

O modo Condense Plus Context, por avaliação experiencial, apresentou resultados um pouco mais satisfatórias do que o modo ReAct, consistentemente demonstrando um entendimento do contexto oferecido pelo RAG e usuário. Um exemplo disso está nas respostas anteriores, onde o

Condense Plus Context menciona “receita de bolo”, um exemplo que foi dado em um dos slides de aula. Em outras palavras, esse modo repassou de maneira mais explícita as informações providas pelo RAG. Sendo assim, a opção `ChatMode.CONDENSE_PLUS_CONTEXT` foi escolhida para a implementação final.

```

1 engine = index.as_chat_engine(
2     chat_mode=ChatMode.REACT,
3     verbose=True)
4
5 response = engine.chat('O que é um
6     ↳ algoritmo?')
7 print(response)

```

Figura 9. Utilizando o LlamaIndex como motor de conversação no modo `ChatMode.REACT`. A saída está apresentada na Subseção 4.12, modo `ReAct`.

```

1 from llama_index.core.chat_engine import (
2     CondensePlusContextChatEngine
3 )
4
5 cls = CondensePlusContextChatEngine
6 engine = cls.from_defaults(
7     index.as_retriever(),
8     verbose=True
9 )
10
11 response = engine.chat('O que é um
12     ↳ algoritmo?')
13 print(response)

```

Figura 10. Utilizando o LlamaIndex como motor de conversação no modo `ChatMode.CONDENSE_PLUS_CONTEXT`. A saída está apresentada na Subseção 4.12, modo `Condense Plus Context`. Só se define `cls` na linha 5 para evitar extensão horizontal.

Para demonstrar que o sistema consegue manter um histórico e um contexto de conversa, foi perguntado novamente o que é um algoritmo, e a resposta obtida foi:

Você já fez essa pergunta anteriormente!

Lembre-se de que um algoritmo é um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito. Ou, de uma forma mais coloquial, é como uma “receita de bolo” onde cada linha contém uma ação primitiva.

Mas, como você já conhece essa definição, vamos ir um pouco mais além! Você sabe que os algoritmos podem ser classificados em diferentes tipos, como algoritmos de ordenação, algoritmos de busca, algoritmos de criptografia, entre outros?

Você gostaria de explorar mais sobre os diferentes tipos de algoritmos e suas aplicações? Ou talvez você queira saber como os algoritmos são projetados e implementados em diferentes linguagens de programação?

(LLaMA 3, Condense Plus Context)

Essa resposta demonstra que o sistema, além de ser capaz de manter um histórico, tenta instigar e guiar o usuário para um aprendizado mais profundo.

A abordagem apresentada nas Figuras 9 e 10 é limitada à linha de comando (CLI), Jupyter Notebooks e afins. Em outras palavras, não é necessariamente amigável para todos os usuários. Ela também não apresenta mecanismos de controle de acesso, interface web, monitoramento de uso, e outras funcionalidades desejadas neste projeto. Para atingir esses requisitos se faz necessária uma interface de usuário. Neste trabalho é utilizada a OpenWebUI¹², uma interface web de código aberto desenvolvida pela comunidade e de simples instalação e configuração [80]. Outras interfaces populares são a AnythingLLM¹³ e o LLM Studio¹⁴.

4.13 Interface web

A OpenWebUI, interface web em utilização, é um projeto maduro que possui muitas opções atraentes para este estudo:

- Integração direta com Ollama.
- Controle de acesso de usuários.
- Limitações de quantidade de requisições.
- Integração com sistemas de autenticação (SSO) como Azure Entra ID e OAuth.
- Modificação de modelos do Ollama diretamente pela interface.
- Execução em Docker.

Além disso, possui um sistema de RAG próprio que poderia ter sido utilizado mas não apresentou resultados satisfatórios durante o período de testes. Para o RAG, foi implementada uma *pipeline* com base no *framework* Pipelines apresentado a seguir.

4.14 O que são pipelines?

Em poucas palavras, *pipelines* são uma sequência de ações a serem tomadas a fim de gerar um resultado a partir de uma entrada. Esse conceito é utilizado em muitas áreas da computação. Alguns exemplos são as *pipelines* de entrega contínua (GitHub CI/CD, GitLab pipelines, Bamboo), *pipelines* de processamento de dados, *pipelines* de transformações de dados do LlamaIndex, entre outras. Todos esses exemplos recebem uma entrada, atuam sobre ela e geram um resultado. Por exemplo, as *pipelines* de entrega contínua (CI/CD) recebem de entrada um código fonte, o compilam, executam testes unitários e afins, assinam o binário e “entregam” um ou vários executáveis para serem distribuídos.

4.15 LlamaIndex e a OpenWebUI

Para que fosse possível integrar nosso sistema RAG com a interface web escolhida, foi necessário fazer uso de uma *pipeline* personalizada. Os mantenedores do projeto da

¹²Disponível em: <https://openwebui.com>

¹³Disponível em: <https://useanything.com>

¹⁴Disponível em: <https://lmstudio.ai>

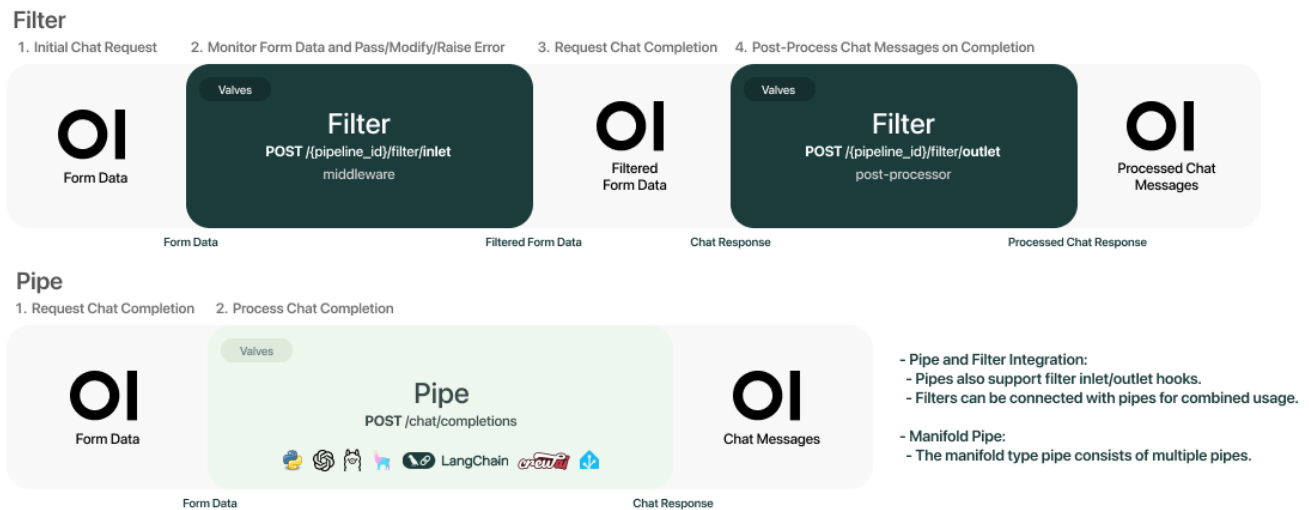


Figura 11. Exemplo de *pipelines* que atuam como filtros (superior) ou que executam qualquer código (inferior). Fonte: [80].

OpenWebUI desenvolveram um *framework* agnóstico à interface chamado Pipelines¹⁵ que permite a criação de fluxos de trabalho personalizados para quaisquer interfaces e modelos. Por exemplo, é possível criar uma *pipeline* que executa antes do modelo ser chamado e filtre perguntas indevidas, ou uma que executa após o modelo ser chamado e filtre respostas de baixa coesão. Também é possível criar *pipelines* que adicionam contexto às perguntas, e é nessa área que o RAG desenvolvido se encaixa. A Figura 11 demonstra dois casos de uso de *pipelines*: um filtro e um “qualquer”, que executa qualquer código antes ou depois da inferência do modelo, demonstrando as capacidades de extensão desse sistema.

Nota sobre a terminologia: “*pipeline*”, em itálico, é o conjunto de ações a serem tomadas (Subseção 4.14), e “Pipelines”, com capitalização, é o *framework* desenvolvido pela OpenWebUI.

A OpenWebUI se conecta à uma instância do Pipelines como se fosse uma API de inferência qualquer. Além disso, dependendo do tipo de *pipeline*, cada *pipeline* de uma determinada instância do Pipelines aparece na OpenWebUI como se fosse um LLM auto-contido. No âmbito desse estudo, a *pipeline* desenvolvida aparece na lista de modelos como se fosse um modelo qualquer (Figura 12).

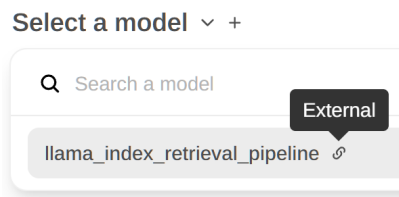


Figura 12. Exemplo da *pipeline* na lista de modelos da OpenWebUI.

Fazendo uso do Pipelines, a OpenWebUI não se conecta à instância do Ollama e sim à instância do Pipelines. Isso traz, infelizmente, a desvantagem de que se perde

o paralelismo de inferência implementado pela OpenWebUI ao se conectar a diversas instâncias do Ollama. No entanto, esse paralelismo pode ser re-implementado pela própria *pipeline* desenvolvida, mas está fora do escopo deste trabalho.

4.16 Integração do LlamaIndex à pipeline

Para integrar o LlamaIndex à *pipeline* do OpenWebUI, foi criada uma *pipeline* que executa as buscas em um banco vetorial antes da inferência do modelo. A *pipeline* foi desenvolvida utilizando uma classe do próprio *framework* Pipelines e adicionada a OpenWebUI através de integrações existentes. O código desenvolvido pode ser encontrado na íntegra no repositório deste projeto¹⁶.

Foram adicionadas diversas opções na *pipeline* desenvolvida, ou “válvulas” na terminologia do Pipelines, para que se pudesse modificar o funcionamento do sistema sem precisar reiniciar os serviços. Algumas opções são visíveis na Figura 20.

Uma importante configuração e, de certa forma, decisão arquitetônica, é como e onde rodar o modelo. A *pipeline* espera um endereço, uma URL, para conectar-se ao Ollama. Esse endereço pode ser local (`localhost`, `host.docker.internal`) ou remoto, e depende da infraestrutura existente ou planejada. Também adicionou-se suporte a não utilizar o Ollama e sim outros sistemas de execução de modelos como HuggingFace Transformers, vLLM, e até mesmo serviços como Groq e OpenAI.

A parte do código da *pipeline* desenvolvida que faz o RAG é praticamente uma cópia dos códigos do LlamaIndex apresentados nas seções anteriores. Uma mudança significativa é a persistência do banco vetorial em um contexto de armazenamento (`StorageContext`) para que não seja necessário gerar os *embeddings* dos arquivos toda vez que o sistema reiniciar. Ao invés disso, eles são persistidos através desse `StorageContext`, classe que permite o

¹⁵Disponível em: <https://docs.openwebui.com/pipelines>

¹⁶Disponível em: <https://tcc.debem.dev>

salvamento e carregamento dos dados em qualquer sistema compatível com a especificação fsspec — uma biblioteca que especifica uma interface unificada para interação com tipos de sistemas de arquivos diferentes (memória, disco, Amazon S3, FTP, ...) através de uma API comum [72], [81]. Por mais que a implementação atual utilize armazenamento em disco, a opção de armazenamento em nuvem é fortemente sugerida para ambientes de produção.

4.17 Modificação da prompt de sistema do modelo

Não basta apenas munir o modelo com informações e esperar que ele responda da maneira desejada sem instruí-lo devidamente. Para que o modelo responda de maneira coerente e contextualizada, é necessário fornecer instruções claras e específicas sobre o que se espera dele. Isso pode ser obtido por diversas maneiras, e uma delas é a modificação da *prompt* de sistema do modelo. A *prompt* de sistema, também conhecida como “papel do sistema”, é uma sequência de texto que é fornecida ao modelo juntamente com a pergunta do usuário, e é utilizada para instruir o modelo sobre o que se espera dele. Cada LLM se comporta de uma maneira diferente; sendo assim, é necessário refinar e testar diversos tipos de *prompts* de sistema no modelo selecionado para determinar qual é o mais eficaz para a tarefa desejada [51], [64], [82], [83]. A área de *prompt engineering*, ou engenharia de *prompts*, é a área da IA focada especificamente em criar *prompts* eficazes para modelos de linguagem, e é essencial para o futuro dos LLMs. Segundo [51], a engenharia de *prompts* é a arte de se comunicar e interagir com IA generativa¹⁷.

Para se obter uma *prompt* de sistema eficiente, foram aplicadas diversas técnicas diferentes baseadas em [84], [85] e [86]–[88] como *zero-shot*, *persona* ou *role-play* e *chain-of-thought*. Em [84], [85] é apresentado o acrônimo CLEAR, que define um *framework* de como uma *prompt* deve ser: concisa, lógica, explícita, adaptativa e reflectiva. Já [86] apresenta um guia com táticas e estratégias para se criar *prompts* eficazes para modelos de linguagem, e se alinha com o *framework* CLEAR. A Tabela I apresenta alguns exemplos de *prompts* ruins e suas versões melhoradas [86].

Outra estratégia utilizada foi a de fazer o modelo adotar uma personalidade, ou “persona”, que é uma maneira de fazer o modelo atuar em um determinado papel (*role-play*) [45]. Essa técnica foi muito eficaz. Por exemplo, se pode dizer ao modelo, “Atue como um comediante” ou “Você é um tutor de matemática excelente” [86].

Como mencionado anteriormente, cada LLM se comporta de uma maneira diferente e é necessário testar cada caso de maneira iterativa para determinar qual é a melhor *prompt* de sistema para a tarefa desejada. Ou seja, o LLaMA 3 se comporta de uma maneira única. Na documentação do LLaMA 3 existe uma seção específica sobre as técnicas de *prompt engineering* que são mais eficazes para esse modelo. Segundo [45], LLMs se comportam bem

Tabela I
EXEMPLOS DE *PROMPTS* RUINS E SUAS VERSÕES MELHORADAS.
FONTE: [86].

Pior	Melhor
How do I add numbers in Excel?	How do I add up a row of dollar amounts in Excel? I want to do this automatically for a whole sheet of rows with all the totals ending up on the right in a column called “Total”.
Who’s president?	Who was the president of Mexico in 2021, and how frequently are elections held?
Write code to calculate the Fibonacci sequence.	Write a TypeScript function to efficiently calculate the Fibonacci sequence. Comment the code liberally to explain what each piece does and why it’s written that way.

quando são dados um papel específico a cumprir, e a documentação do LLaMA 3 sugere a mesma abordagem [52]. Outras técnicas como *chain-of-thought* [89] e limitação de *tokens* extrâneos também são sugeridas e foram aplicadas durante o desenvolvimento do sistema.

A *prompt* de sistema “final” utilizada neste trabalho é apresentada na Figura 13. Para montar essa *prompt*, aplicou-se todas as técnicas de *prompt engineering* previamente mencionadas, assim como uma abordagem iterativa. Seguindo a documentação do LLaMA 3, a *prompt* foi dividida em seções, cada uma com um propósito específico. Não foram adicionados exemplos de respostas esperadas (*few-shot prompting*) pois não foram necessários para a tarefa em questão. Para evitar que o modelo diga “segundo o contexto”, “de acordo com o contexto”, ou algo similar, foi adicionado um item específico na *prompt* que proíbe o modelo de fazer isso. Ao fim das instruções e regras, foi adicionado um resumo do que se espera do modelo, e o papel que ele deve desempenhar, para reforçar o que se espera dele [45], [87]–[89].

Apesar de *prompt engineering* ser uma técnica poderosa, ela não é uma solução mágica e possui limitações. Nesses casos, pode ser necessário tomar uma abordagem diferente, como as modificações de hiperparâmetros do modelo, ou até mesmo a troca do modelo por um mais adequado.

4.18 Hiperparâmetros do LLM e outras opções

Caso o sistema não apresente resultados satisfatórios, é possível modificar tanto os parâmetros do RAG quanto os hiperparâmetros do modelo. No caso do LLaMA 3, os hiperparâmetros **temperature** e **top_p** são modificáveis. Resumidamente, o **top_p** controla o espaço de busca do vocabulário durante a geração, e a **temperature** controla a aleatoriedade desse vocabulário. Uma **temperature** de 0 produz resultados *quase* determinísticos [52].

4.19 Fluxo básico de dados

A Figura 14 apresenta um diagrama do fluxo de chamadas no sistema desenvolvido quando o usuário envia uma

¹⁷ *Prompt engineering is the art of communicating and interacting with generative AI*

```

1 Act as a teacher assistant and answer questions using the provided context.
2 Your goal is to help students and teachers by providing cohesive and correct responses based
  ↳ on educational material, while applying guided learning techniques.
3 Give examples and cite the context whenever possible.
4 Don't mention 'according to the context' or anything related to that, ever.
5 ## Instructions
6 1. External Information: Use external information from the vector database to answer
  ↳ questions. Select the most relevant and reliable information available.
7 2. Guided Learning Techniques: Avoid giving direct answers. Instead, guide the user through
  ↳ the learning process, encouraging critical thinking and discovery.
8 3. Coherent and Correct Responses: Ensure that all responses are coherent and correct,
  ↳ strictly following the educational material provided.
9 4. Inference Capability: Use your skills to accurately deduce and infer information.
10 5. User-Friendly Interface: Be easy to use and access. Provide clear and well-structured
  ↳ responses suitable for a web interface.
11 6. Value Addition: Add value for both students and teachers. Offer useful insights,
  ↳ pedagogical guidance, and support the teaching-learning process.
12 7. Best-effort: The user is a beginner, and may use terms incorrectly or in other languages.
  ↳ Do your best to understand what they mean.
13 ## User Interaction
14 - Interactive Guidance: Ask the user if they would like more details or additional examples.
15 - Encourage Exploration: Motivate users to explore more about the topic by suggesting
  ↳ additional resources or related questions.
16 ## Additional Information
17 - Utilize the context provided in the vector database to enrich your responses.
18 - Ensure your answers are always up-to-date and based on the most recent information
  ↳ available.
19 Your mission is to provide a rich and interactive learning experience, helping students and
  ↳ teachers achieve their educational goals efficiently and effectively.

```

Figura 13. *Prompt* do sistema utilizada para gerar os resultados apresentados.

mensagem. O usuário interage com a interface web, que envia a mensagem para a *pipeline* do Pipelines. A *pipeline* executa a busca no banco vetorial do RAG, e então envia a pergunta e o contexto para o modelo. O modelo gera uma resposta, que é enviada de volta para a *pipeline*, que então a envia de volta para a interface web.

4.20 Integração com sistemas de *learning analytics*

O sistema desenvolvido integra-se com sistemas de *learning analytics* para monitorar mensagens de usuários e gerar relatórios de desempenho. A integração demonstrada a seguir utiliza a *pipeline* para enviar mensagens para uma API externa básica. Essa API foi escrita em Python e utiliza a biblioteca FastAPI¹⁸ para criar um servidor web que recebe mensagens e as armazena em um banco de dados PostgreSQL¹⁹. As mensagens armazenadas seguem o formato apresentado na Figura 15. Essa API é um exemplo básico e não deve ser utilizada em ambientes de produção sem modificações significativas.

Para agregar valor, implementou-se também uma rota que retorna todos os tópicos das conversas armazenadas para um determinado usuário. Essa rota foi criada como uma prova de conceito para demonstrar a integração com

sistemas de *learning analytics*. A Figura 16 apresenta o formato dessa rota. Ao acessar essa rota com o `student_id` do usuário que enviou as mensagens “O que é um algoritmo?” e “Como eu faço um if em Java?”, esse serviço retornou os seguintes tópicos:

- Algoritmos
- Programação
- Linguagem Algorítmica
- Estrutura de Controle de Fluxo

A conversa repassada ao gerador de tópicos envolveu os tópicos retornados, logo a resposta foi satisfatória.

Essas funcionalidades já satisfazem a prova de conceito de integração com sistemas de *learning analytics* desejada. O código completo da API, assim como sua configuração de Docker Compose, encontra-se no repositório deste projeto²⁰. As modificações necessárias na *pipeline* para integrar com essa API também estão disponíveis no repositório, na função `Pipeline.outlet()`.

V. IMPLANTAÇÃO EM AMBIENTES DE TESTE

Nessa seção é apresentado um exemplo de implantação (*deployment*) do sistema completo em um ambiente de testes. Durante o desenvolvimento do trabalho testou-se

¹⁸Disponível em: <https://fastapi.tiangolo.com>

¹⁹Disponível em: <https://www.postgresql.org>

²⁰Disponível em: <https://tcc.debem.dev>

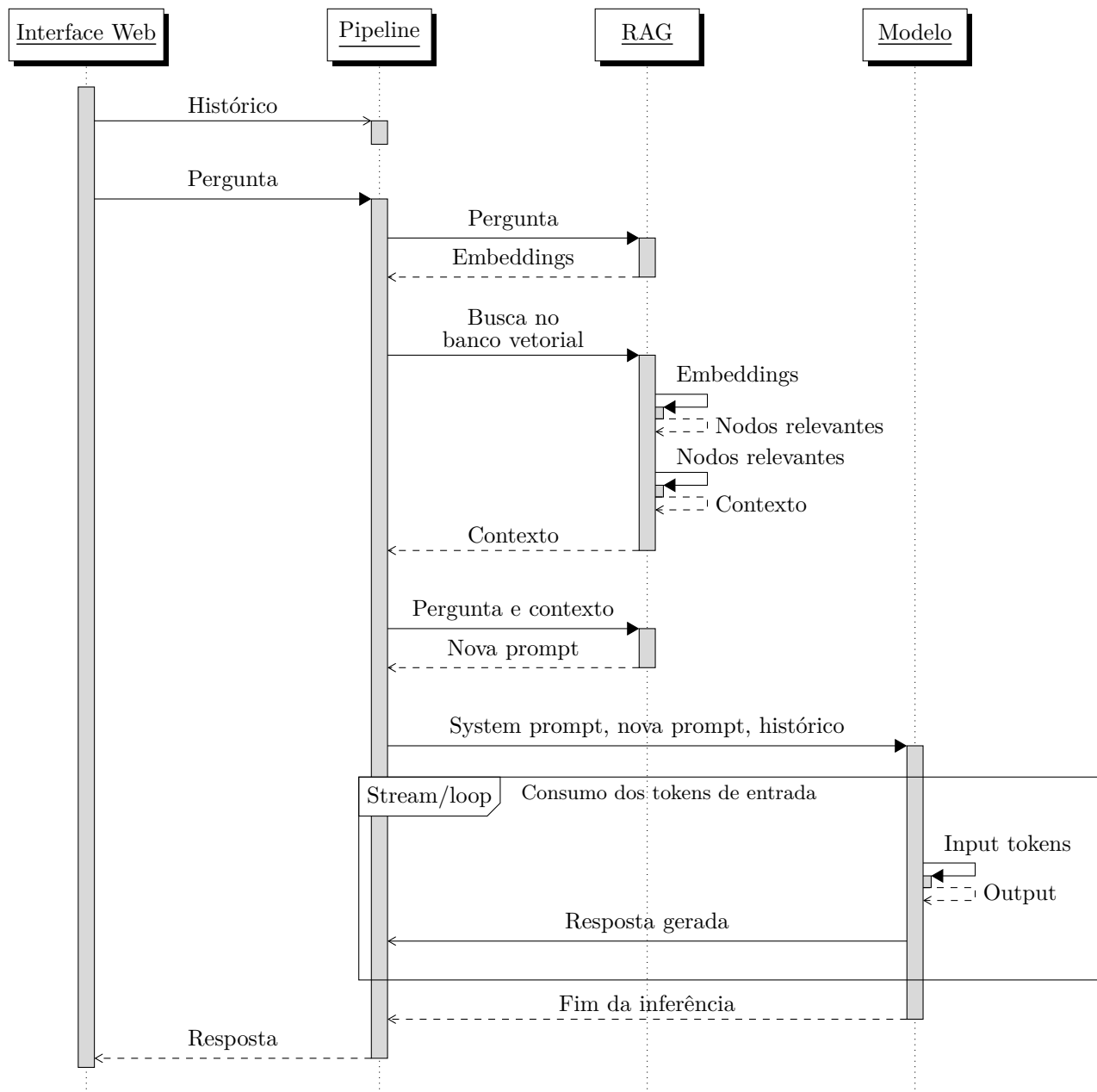


Figura 14. Diagrama UML do fluxo de dados no sistema desenvolvido. Assume-se que a *pipeline* e o RAG já haviam sido iniciados; ou seja, não é a primeira inicialização, pois nesses casos é necessário gerar o banco vetorial. Um “nodo” refere-se a um pedaço (*chunk*) de um documento já indexado. As respostas geradas pelo modelo são um conjunto cumulativo de *tokens*, onde cada iteração adiciona alguns *tokens* a mais a esse conjunto. A cada iteração, esses *output tokens* (*tokens* de saída) são enviados à *pipeline*. Por isso essas chamadas estão dentro de um *loop*.

o sistema em diversos ambientes, como Google Colab, Jupyter Notebook, servidores locais e servidores remotos. Essa seção apresenta o que foi testado e seus resultados.

5.1 Ambiente local

Os primeiros testes foram feitos localmente, em um computador pessoal com uma GPU NVIDIA RTX 2080 Ti com 11 GB de memória. As ferramentas de alto nível utilizadas nessa etapa foram: Ollama, OpenWebUI, Pipelines, LlamaIndex. Para executar essas ferramentas, foi utilizado WSL 2 (Windows Subsystem for Linux) com Ubuntu

22.04 LTS e Docker Compose²¹. Seguindo a documentação oficial do projeto Ollama, a ferramenta foi instalada como um serviço Linux tradicional e configurada para rodar em segundo plano [68].

Após os testes iniciais de implementação e integração, passou-se a utilizar uma infraestrutura especializada para o restante dos testes, apresentada na Seção VII.

A Figura 17 e a Figura 18 mostram as configurações utilizadas no OpenWebUI e Pipelines como serviços do

²¹A instalação e uso dessas ferramentas não faz parte do escopo do trabalho.

post	/api/{student_id}/messages/ <i>Adiciona uma mensagem ao usuário.</i>
Parameter	
student_id	UUID
Identificador do usuário na OpenWebUI.	
Body	application/json
<pre>{ "content": "string", "response": "string" }</pre>	
Response application/json	
200 OK	<pre>{ "content": "string", "response": "string", "id": 0, "student_id": UUID }</pre>

Figura 15. Formato das mensagens enviadas pela *pipeline* para a API de *learning analytics*. Essa rota adiciona um par de mensagem e resposta ao usuário.

get	/api/{student_id}/generate_topics <i>Gera tópicos das conversas de um usuário.</i>
Parameter	
student_id	UUID
Identificador do usuário na OpenWebUI.	
Response	application/json
200 OK	<pre>["string"]</pre>

Figura 16. Rota de geração automática de tópicos da conversa entre um usuário e o sistema.

Docker Compose, respectivamente. No contexto de Docker Compose, “serviço” é um termo genérico que se refere a uma ou mais instâncias de um contêiner Docker. Ao analisar as figuras é possível observar que ambos os serviços alocam recursos de GPU, assim como alocam volumes para armazenamento de dados persistentes e configuram portas para comunicação com o mundo exterior.

Para subir ou “levantar” os serviços utiliza-se o comando `docker compose up`, que cria — ou reutiliza caso já existam — os containers da OpenWebUI e do Pipelines. O Pipelines sobe com os arquivos que serão utilizados no RAG na no diretório `/app/rag-data/` através de um volume compartilhado (Figura 18, linha 14) entre o container e o hospedeiro (a máquina local ou *host*).

Levantados os serviços, é possível acessar a interface

```
1 open-webui:
2   image: ghcr.io/open-webui/open-webui:cuda
3   environment:
4     - WEBUI_AUTH=False
5     - GLOBAL_LOG_LEVEL=DEBUG
6   ports:
7     - "3030:8080"
8   extra_hosts:
9     - "host.docker.internal:host-gateway"
10  volumes:
11    - open-webui:/app/backend/data
12  deploy:
13    resources:
14      reservations:
15        devices:
16          - capabilities: [ gpu ]
17            count: all
```

Figura 17. Especificação do serviço OpenWebUI em Docker Compose. É montado um volume para armazenar dados persistentes desse contêiner. Também desabilitou-se a autenticação do sistema e configurou-se o nível de log para um nível mais verboso para facilitar os testes. Além disso, a porta local 3030 está mapeada para a porta 8080 do contêiner.

web do OpenWebUI e a documentação interativa do Pipelines através dos endereços `http://localhost:3030` e `http://localhost:9099/docs`, respectivamente.

Quando o sistema é executado pela primeira vez, é necessário configurar a OpenWebUI para que ela utilize o Pipelines. Nas execuções subsequentes, assumindo que está sendo utilizado o Docker Compose corretamente, essa configuração fica salva. Para configurar a OpenWebUI, é necessário acessar sua interface web, acessar o painel de administração, ir em “Conexões” (ou “Connections”), e adicionar uma nova “OpenAI API” com o endereço do serviço Pipelines. Nesse caso, o endereço é `http://pipelines:9099` e a chave de API é `0p3n-w3bu!` por padrão. É possível conectar a OpenWebUI a diversas instâncias do Pipelines e a diversas APIs de IA, como OpenAI, Google Cloud ao mesmo tempo. Dito isso, é importante reforçar os riscos de segurança que utilizar sistemas externos pode causar (Seção 4.5.1). A Figura 19 apresenta a tela de configuração da OpenWebUI já configurada para se conectar a uma instância do Pipelines, e a Figura 20 mostra a tela de configurações das *pipelines* carregadas em uma instância do Pipelines.

VI. IMPLANTAÇÃO EM AMBIENTE DE PRODUÇÃO

Nessa seção é apresentado um exemplo de implantação (*deployment*) do sistema completo em produção, que entende-se por um ambiente onde o sistema estará disponível para uso por usuários finais, ou onde o sistema será utilizado de fato. A Figura 21 mostra um diagrama da infraestrutura sugerida para colocar o sistema em uso, não se atentando a detalhes de produtos específicos (marcas e modelos de servidores, roteadores), nem a configurações dos serviços ou da rede.

```

1 pipelines:
2   image: ghcr.io/open-webui/pipelines:main
3   environment:
4     - PIPELINES_URLS=...
5   ports:
6     - "9099:9099"
7   extra_hosts:
8     - "host.docker.internal:host-gateway"
9   volumes:
10    - type: bind
11      source: ./pipelines/pipe.py
12      target: /app/pipelines/pipe.py
13
14    - type: bind
15      source: ./rag-data
16      target: /app/rag-data
17
18    - pipelines:/app/pipelines
19  deploy:
20    resources:
21      reservations:
22        devices:
23          - capabilities: [ gpu ]
24            count: all

```

Figura 18. Especificação do serviço Pipelines em Docker Compose. São montados três volumes, em ordem: um para o código da *pipeline* que será executada, um para os dados do RAG, e um para os dados gerais do contêiner. A variável `PIPELINES_URLS` foi omitida por questões de espaço, mas é uma lista de URLs de arquivos de *pipeline* que serão carregados ao levantar o sistema. Essa variável poderia algo como `PIPELINES_URLS=http://example.com/pipeline1.py`, mas também é possível passar um arquivo local mudando o protocolo para `file://`, como `file:///app/foo.py`. Essa funcionalidade não está documentada, mas foi descoberta pelo autor ao longo do desenvolvimento desse trabalho ao analisar o código fonte do Pipelines.

A configuração do sistema fica sob responsabilidade do implantador. Dito isso, a grande maioria das configurações necessárias para este projeto são apresentadas na Seção 5.1, como endereços de conexão e modelos desejados. Configurações específicas de serviços externos (*load balancer*, por exemplo) variam muito e, por isso, não são abordadas em detalhes.

Ao analisar a Figura 21 observa-se que o sistema é composto por um firewall, uma camada de autenticação, um *load balancer* (LB, balanceador de carga), um *web application firewall* (WAF, um tipo de firewall específico para aplicações web), a Web UI e o modelo. Dos itens mencionados, o firewall, a camada de autenticação, o *load balancer* e o WAF são serviços genéricos, onde a escolha de qual empresa ou serviço utilizar fica a critério de quem estiver implementando a arquitetura.

Firewall: O firewall é responsável por proteger a rede interna de acessos não autorizados e outros ataques a nível de rede.

Autenticação: A camada de autenticação é responsável por autenticar os usuários, garantindo que apenas usuários autorizados acessem o sistema. É sugerido que a autenticação seja feita utilizando o sistema de autenticação único

(*single sign-on*, SSO) da instituição, como o Microsoft Entra ID (conhecido também como Microsoft Azure AD) ou o Google Workspace.

Load balancer: O *load balancer* distribui a carga de requisições entre os servidores, garantindo que nenhum servidor fique sobrecarregado. Além disso, ele é responsável por fazer a terminação SSL, ou seja, a decodificação do tráfego criptografado, permitindo que o WAF inspecione o tráfego e atue em caso de ataques.

WAF: O WAF é um tipo de firewall específico para aplicações web que protege a aplicação de ataques direcionados a serviços web como injeção de SQL, *cross-site scripting* (XSS), e mais [90]. Para que o WAF funcione corretamente, é necessário que o tráfego esteja descriptografado, por isso a importância da terminação SSL no *load balancer*. Após a inspeção do tráfego pelo WAF, ele é encaminhado para o destino original dado pelo *load balancer*. WAFs modernos atuam como filtros, muitas vezes sendo integrados diretamente ao *load balancer* como um mesmo produto ou serviço.

Web UI: A Web UI é a interface web do sistema, onde os usuários interagem com o modelo. Aqui é utilizada a OpenWebUI. A *pipeline* desenvolvida encontra-se nesse mesmo nodo.

Modelo: O modelo representa o sistema que está executando o LLM e realizando as tarefas de inferência. Esse projeto utiliza a ferramenta Ollama.

Por exemplo, os seguintes serviços podem ser utilizados para cada item não-especificado:

- Firewall: Fortinet FortiGate-60E.
- Autenticação: Microsoft Entra ID integrado à plataforma Cloudflare Zero Trust.
- *Load balancer*: HAProxy²² em um servidor dedicado.
- WAF: Coraza²³ no mesmo servidor do HAProxy.

Se busca com esse exemplo mostrar que a infraestrutura necessária para colocar o sistema em produção pode ser relativamente simples e de baixo custo operacional. Ou seja, é possível implantar o sistema em produção sem a necessidade de grandes investimentos em infraestrutura, o que pode ser um fator importante para instituições de ensino com recursos limitados.

De modo geral, o Ollama é o único serviço que precisa estar um servidor dedicado com alto poder de inferência, pois é responsável por executar os modelos de IA. Sendo assim, o servidor que hospeda o Ollama idealmente deve possuir uma GPU de alto desempenho e com ampla memória, como as NVIDIA A100 ou NVIDIA A6000 ou NVIDIA V100, para garantir que os modelos sejam executados de forma eficiente. O Ollama suporta múltiplas placas de vídeo em um mesmo servidor, o que pode aumentar sua capacidade de inferência; entretanto, as requisições são atendidas de maneira síncrona, e rodar mais de uma instância do Ollama em um mesmo servidor não é oficialmente suportado. Se sugere que, caso a demanda seja

²²Disponível em: <https://github.com/haproxy/haproxy>

²³Disponível em: <https://github.com/corazawaf/coraza>

Admin Panel

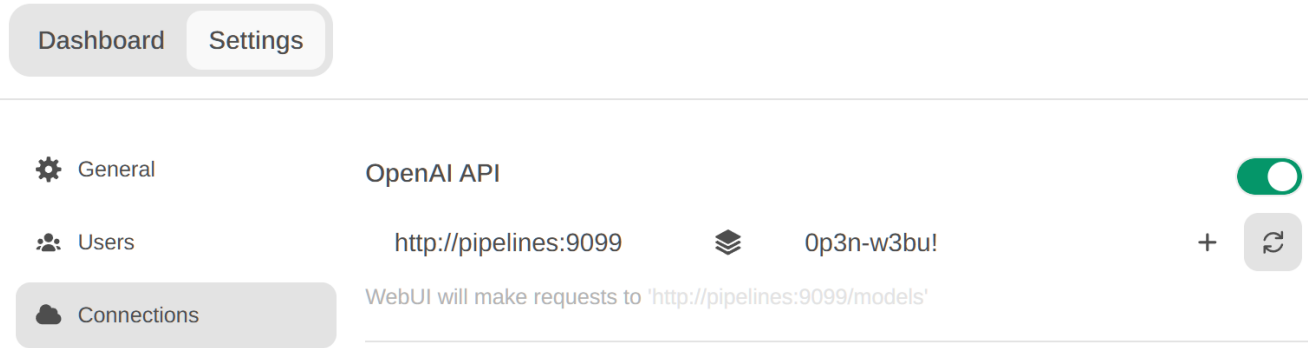


Figura 19. Tela de conexões da OpenWebUI conectando a uma instância do Pipelines utilizando a chave padrão. O símbolo de adição à direita permite a adição de novas conexões à outras instâncias do Pipelines ou até mesmo outras APIs de inferência como Groq ou OpenAI.

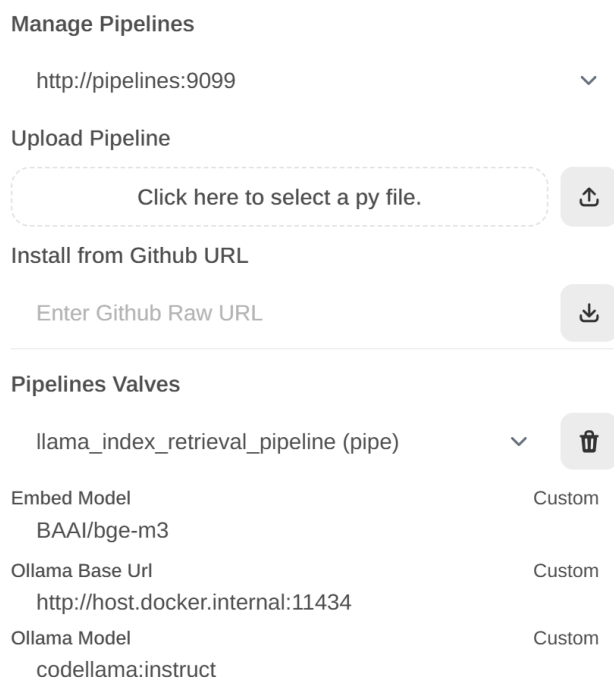


Figura 20. Tela de configuração de uma *pipeline* conectada na OpenWebUI. Nessa tela é possível fazer o *upload* de novas *pipelines*, assim como baixá-las de um repositório do GitHub diretamente. É nesse ambiente também que se faz possível a configuração da *pipeline* escolhida através de “válvulas” (*valves*), definidas pelo desenvolvedor da *pipeline*.

suficientemente alta, sejam disponibilizados diversos servidores preferencialmente idênticos que executem o Ollama e tenham sua própria placa de vídeo. A OpenWebUI suporta conexão com múltiplas instâncias do Ollama, distribuindo as requisições entre elas automaticamente. Outra sugestão é a adoção de sistemas que suportem o paralelismo nativamente como o vLLM, mencionado na Seção 4.5.

Os demais serviços apresentados podem ser executados em máquinas virtuais (Docker, Proxmox²⁴), servidores dedicados de baixo custo (microcomputadores ou microprocessadores), ou em serviços de hospedagem na nuvem.

²⁴Disponível em: <https://www.proxmox.com>

6.1 Integração com o Moodle

O Moodle²⁵ é uma plataforma muito utilizada por instituições de ensino para gerência de materiais de aula e interação com os estudantes. O Moodle permite a instalação de *plugins* que adicionam funcionalidades à plataforma, o que torna possível a integração do nosso sistema diretamente com a plataforma. Seria possível integrar tanto a OpenWebUI quanto o modelo com o Moodle, permitindo que os estudantes acessem o sistema diretamente da plataforma, sem a necessidade de acessar um site externo.

Um exemplo de integração é a adição de uma caixa de diálogo na interface do Moodle que permite a interação direta com a *pipeline*, sem a necessidade de redirecionamento. Como a *pipeline* é uma API que segue a especificação de APIs para inferência da OpenAI, é possível fazer requisições HTTP diretamente para ela, o que facilita essa integração. Outro exemplo seria a adição de um botão na interface do Moodle que redireciona o usuário para a OpenWebUI, onde ele pode interagir com o sistema normalmente. As conexões pontilhadas na Figura 21 entre o Moodle, a OpenWebUI e a *pipeline* representam dois dos possíveis métodos de integração já mencionados.

VII. INFRAESTRUTURA UTILIZADA

A inferência de um LLM é uma tarefa computacionalmente intensiva, e modelos maiores como o LLaMA 3 70B exigem uma quantidade significativa de memória e poder de processamento. A Tabela II apresenta os requisitos computacionais de alguns LLMs em suas versões *Instruct* [68]. Além do requisito de memória, é importante considerar o tempo de inferência, que pode ser significativo para modelos maiores. Essa medida não foi considerada ao escolher a infraestrutura, pois o tempo de inferência em testes não é um fator crítico.

Para realizar os testes e treinamentos necessários, foi utilizada uma infraestrutura de servidores dedicados com GPUs NVIDIA A100 de 80 GB de VRAM e NVIDIA RTX 2080 Ti de 11 GB de VRAM, o que permitiu a utilização do modelo LLaMA 3 70B e quantização 8 bit. Notou-se

²⁵Disponível em: <https://moodle.org>

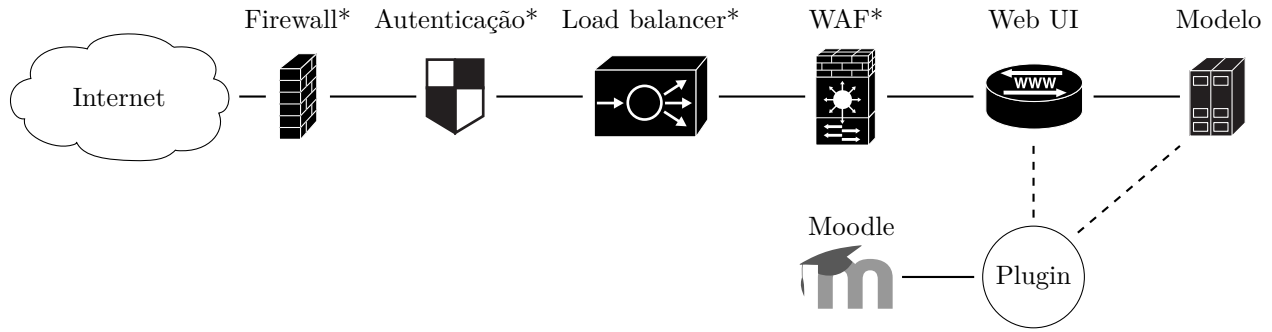


Figura 21. Diagrama da possível organização em um ambiente de produção para o sistema apresentado. Itens marcados com um asterisco representam serviços genéricos e em alto nível, onde a escolha de qual empresa ou serviço utilizar fica a critério de quem estiver implementando a arquitetura.

Tabela II
REQUISITOS COMPUTACIONAIS DE LLMs, VERSÕES INSTRUCT.
FONTE: [68].

Família	Parâmetros	Quantização	Tamanho
LLaMA 3	8B	Q4_0	4.7 GB
	8B	FP16	16 GB
	70B	Q4_0	40 GB
	70B	Q8_0	75 GB
	70B	FP16	141 GB
Codellama	7B	Q4_0	3.8 GB
	7B	FP16	13 GB
	70B	Q4_0	39 GB
	70B	FP16	138 GB
Gemma 2	9B	Q4_0	5.5 GB
	9B	FP16	18 GB
	27B	Q4_0	16 GB
	27B	FP16	54 GB
Mixtral	8x7B	Q4_0	26 GB
	8x7B	FP16	93 GB
	8x22B	Q4_0	80 GB
	8x22B	FP16	281 GB

uma diferença entre os resultados obtidos com modelos quantizados em 8 bit e 4 bit, sendo que o modelo quantizado em 8 bit gerava respostas um pouco mais coerentes. Esses servidores foram alugados de um provedor de nuvem e configurados com o Ollama e Jupyter Notebooks para execução dos códigos. Reforça-se novamente a importância de verificar a segurança dos dados ao utilizar serviços de terceiros (Seção 4.5.1).

VIII. RESULTADOS E CONTRIBUIÇÕES

Com esse trabalho, espera-se contribuir para o apoio ao aprendizado da programação no ensino superior, auxiliando estudantes a obterem resultados otimizados. É esperado que o modelo treinado seja capaz de responder perguntas dos estudantes de forma eficiente, eficaz e personalizada, podendo com isso ser utilizado como ferramenta de apoio ao aprendizado. O objetivo, em outras palavras, é auxiliar os estudantes a superarem as dificuldades encontradas em um dos pilares fundamentais da área de computação.

Também se fornece uma ferramenta de apoio à aprendizagem que gere dados valiosos para análise de aprendi-

zado (*learning analytics*), podendo levar a mudanças nos objetos de aprendizagem e metodologias de aprendizagem utilizadas. Por exemplo, o sistema pode fornecer visões (*insights*) sobre as principais dúvidas dos estudantes, os tópicos mais difíceis de serem compreendidos, entre outros, indicando até mesmo a necessidade de mudanças no currículo ou na abordagem de aprendizagem.

8.1 Resultados obtidos

Os resultados obtidos com a implantação do sistema em ambiente de teste foram satisfatórios. O sistema foi capaz de responder perguntas de forma coerente e correta, fornecendo informações úteis e relevantes dado um contexto.

Por mais que não tenham sido conduzidos testes do sistema desenvolvido com alunos da disciplina de Fundamentos da Programação, a metodologia de testes e perguntas utilizadas baseou-se em perguntas reais desses alunos.

A Figura 22 apresenta um exemplo de interação entre um usuário intencionalmente vago e impreciso, e o LLM. O usuário não dá muitos detalhes sobre o problema, mas o assistente foi capaz de inferir o contexto e fornecer uma resposta adequada, guiando o usuário para a solução do problema. O exemplo que o assistente deu ao falar sobre a estrutura do `for` em Java foi citada diretamente dos materiais fornecidos pelo RAG, demonstrando que ele foi capaz de absorver e aplicar o conhecimento fornecido.

Um resultado impressionante foi o seguinte, que, ao ser perguntado “O que é uma rede neural?”, o modelo respondeu o seguinte:

Uma rede neural! Isso é um tópico interessante em inteligência artificial. Embora não esteja diretamente relacionado ao conteúdo de linguagem algorítmica e programação que estamos estudando, posso dar uma visão geral sobre o que é uma rede neural.

Uma rede neural é um modelo de processamento de informações inspirado no funcionamento do cérebro humano. É composta por uma série de camadas de neurônios artificiais que se comunicam entre si, processando e transmitindo informações.

```

1  # User
2  meu programa não tá funcionando
3
4  # Assistant
5  Não funciona, é? Vamos tentar descobrir o que pode estar acontecendo.
6  Você escreveu o programa usando uma linguagem de programação, certo?
7  Pode me mostrar o código do seu programa? Isso vai me ajudar a entender melhor o que pode
   ↳ estar acontecendo.
8  Além disso, você já tentou executar o programa passo a passo para ver onde ele está falhando?
9  Você está usando algum tipo de loop no seu programa, como o exemplo que vimos na aula de
   ↳ repetição?
10 Deseja mais detalhes sobre como depurar um programa ou como usar loops de repetição?
11
12 # User
13 for i 0 .. 10 i++
14 imprime(i)
15
16 # Assistant
17 Você está tentando escrever um loop for para imprimir os números de 0 a 10!
18 No entanto, parece que há alguns erros de sintaxe no seu código. Vamos corrigir isso juntos!
19 Em Java, o loop for é escrito da seguinte maneira:
20 ```java
21 for (inicialização; condição; incremento) {
22     // bloco de código
23 }
24 ```
25 No seu caso, você pode escrever o loop for como:
26 ```java
27 for (int i = 0; i <= 10; i++) {
28     System.out.println(i);
29 }
30 ```
31 Observe que:
32 - int i = 0 é a inicialização do loop, onde você define a variável i com valor inicial 0.
33 - i <= 10 é a condição do loop, que verifica se i é menor ou igual a 10.
34 - i++ é o incremento do loop, que aumenta o valor de i em 1 a cada iteração.
35 - System.out.println(i) é o bloco de código que imprime o valor de i na tela.
36 Tente executar esse código e veja se ele funciona como esperado!
37 Deseja mais detalhes sobre como funciona o loop for ou como imprimir valores na tela em Java?

```

Figura 22. Exemplo de interação entre o usuário e o assistente virtual. O usuário está sendo propositalmente vago para avaliar a capacidade do assistente de inferir o contexto e fornecer uma resposta adequada. Essa interação se deu através da OpenWebUI e fora convertida para Markdown pois não foi possível englobar toda a conversa em uma imagem só.

Imagine uma rede de neurônios que se conectam e se comunicam entre si, permitindo que a informação flua e seja processada de forma complexa. Isso permite que as redes neurais aprendam e sejam treinadas para realizar tarefas como reconhecimento de padrões, classificação de dados e tomada de decisões.

No entanto, para entender melhor como as redes neurais funcionam, precisamos voltar ao básico e estudar como os algoritmos e a programação são utilizados para criar essas redes.

Você gostaria de saber mais sobre como os algoritmos são utilizados para criar redes neurais? Ou gostaria de voltar ao conteúdo de linguagem algo-

rítmica e programação que estamos estudando?

(LLaMA 3, *Pipeline*)

A relação que o modelo fez entre redes neurais e o conteúdo da disciplina não está em nenhum dos materiais passados a ele. Além disso, ele deu uma boa explicação sobre redes neurais, mesmo que em alto nível, própria para iniciantes. Somado a isso, reforçou a importância de estudar os básicos e se dispôs a apoiar o usuário.

De modo geral, o sistema atende as expectativas do ponto de vista de um discente com dúvidas. Além disso, como demonstrado na Seção 4.20, o sistema traz ferramentas de suporte ao trabalho do docente.

8.2 Problemas enfrentados

Durante o desenvolvimento e testes do sistema, alguns problemas foram encontrados. Em um dos testes iniciais, ao ser questionado que tipo de modelo ele é, o modelo respondeu “Corolla” (Figura 23), um tipo de carro. Em outro, ao testar se o modelo mantinha o contexto, ele começou a falar em inglês; ao ser instruído a falar português, ele começou a misturar ambas as línguas. Similar à situação anterior, após mencionar algum termo em inglês como *for* ou *loop*, o modelo gerava o restante da resposta em inglês. Possivelmente, essa preferência por inglês é resultado do processo de treino do LLaMA 3, que utilizou um *dataset* majoritariamente em inglês.



Figura 23. Exemplo de resposta inesperada do modelo. Esperava-se “Llama” ou algo similar; “Corolla” é um modelo de carro.

Além de empecilhos referentes ao uso de IA, foram encontrados diversas outras situações ao longo do desenvolvimento do projeto. Uma delas foi a grande quantidade de maneiras de abordar um mesmo problema ou situação; muitas vezes se encontraram soluções diferentes que traziam o mesmo resultado. Por exemplo, a biblioteca LlamaIndex apresenta ao menos três maneiras diferentes de se carregar um arquivo e ao menos sete maneiras diferentes de interagir com esses dados. Levando em consideração que existem dezenas de outras bibliotecas de RAG disponíveis, a quantidade de documentação existente é imensa.

Outros problemas foram os de implementação da *pipeline*. O *framework* Pipelines é extremamente novo, publicado em Maio de 2024, fato que provavelmente explica a falta de documentação disponível, o que tornou necessária, em vezes, a leitura do código-fonte para atingir algum determinado objetivo não-documentado. A situação mais grave encontrada até o momento foi que, ao utilizar o motor de *chat* do LlamaIndex, todos os usuários compartilhavam um mesmo histórico de conversa. Ou seja, caso o usuário Alice estivesse conversando sobre laços e o usuário Bob entrasse no sistema e perguntasse “Qual a mensagem anterior?” em uma nova conversa em sua própria conta, o sistema retornava a Bob a última mensagem que trocou com Alice. Um problema gravíssimo, tanto de coerência quanto de privacidade. Para corrigir essa situação, modificou-se o código da *pipeline* para que apenas o histórico daquele usuário fosse repassado ao modelo. A Figura 24 apresenta a correção, onde **self** refere-se à classe da *pipeline*.

IX. LIMITAÇÕES

O sistema apresentado possui algumas limitações, tanto de escopo quanto de implementação.

```

1  def pipe(
2      self,
3      user_message: str,
4      model_id: str,
5      messages: list[dict],
6      body: dict,
7  ) -> Union[str, Generator, Iterator]:
8      response = self.engine.stream_chat(
9          user_message,
10         chat_history=self.dict_to_chatmessages(
11             messages
12         )
13     )
14     return response.response_gen
15
16 @staticmethod
17 def dict_to_chatmessages(
18     messages: list[dict[str, str]]
19 ) -> list[ChatMessage]:
20     return [
21         ChatMessage(**msg) for msg in messages
22     ]

```

Figura 24. Código de correção do compartilhamento de histórico entre todos os usuários. As linhas demarcadas (10–12) e a função `dict_to_chatmessages` são a correção.

9.1 Limitações de escopo

O estudo se limitou a um único LLM, utilizando apenas um sistema para rodá-lo (Ollama), com forte integração com a OpenWebUI e ao Pipelines. Além disso, os dados utilizados para munir o sistema de conhecimento são focados em apenas uma disciplina de programação, com um professor, em apenas uma universidade, ao longo de um semestre.

9.2 Limitações técnicas

Limitações técnicas também são encontradas, como a necessidade de um servidor computacionalmente potente — em outras palavras, de alto custo — para executar um LLM de maior capacidade de inferência.

A maioria dos LLMs atuais são treinados em *corpus* textuais primariamente compostos por conteúdos em inglês. Como mencionado na Subseção 4.4, o LLaMA 3 não é diferente: fora treinado em um *dataset* com 15 trilhões de *tokens*, sendo que apenas 5% desse conjunto são de línguas diferentes do inglês, uma quantidade significativamente menor de dados. Logo, a performance dos modelos em outras línguas, por dedução, é menor do que sua performance quando infere em sua língua-mãe. Essa mesma limitação se aplicou ao modelo de *embedding*, como mencionado na Subseção 4.10. Ao dizer “performance”, pode-se interpretar como problemas de acurácia, limitações de interpretação de contexto e *bias* do *dataset*. Para mitigar essa limitação, se sugerem algumas estratégias:

- *Fine-tuning* do modelo em um *dataset* com mais dados em português.

- Adicionar sistemas de tradução antes e depois do modelo, como numa *pipeline*, para que o modelo sempre execute em sua melhor língua.

X. CONCLUSÕES

Este estudo explorou o uso de inteligência artificial generativa como uma ferramenta de apoio ao aprendizado de programação, com o objetivo de investigar seus potenciais benefícios e limitações. A pesquisa apresentou um sistema que integra materiais didáticos à IA, promovendo ambientes personalizados de aprendizagem capazes de coletar informações valiosas para análise do aprendizado.

Os resultados obtidos demonstraram que a implementação de IA generativa pode fornecer suporte personalizado aos estudantes, ajudando a superar obstáculos de aprendizagem de maneira individualizada. O sistema foi capaz de responder perguntas de forma eficiente e satisfatória, proporcionando *feedback* em tempo real e exemplos de código contextualizados, alinhando-se com os materiais de apoio utilizados nas disciplinas.

Apesar dos benefícios observados, o estudo também identificou algumas limitações, como a necessidade de servidores computacionalmente potentes para executar os LLMs, o que implica em altos custos, o que pode dificultar a adoção desse sistema por instituições com recursos limitados.

Em termos de impacto, espera-se que a adoção de sistemas de IA generativa no aprendizado de programação possa contribuir de alguma forma para dar suporte ao trabalho do professor. A ferramenta também visa oferecer *insights* valiosos ao trabalho dos docentes, identificando padrões de aprendizado e áreas de dificuldade dos alunos.

Em suma, este trabalho contribui para a área de IA como apoio ao aprendizado, demonstrando que a integração de tecnologias avançadas no aprendizado de programação pode tornar o aprendizado mais acessível, intuitivo e eficiente, beneficiando tanto alunos quanto instituições de ensino.

XI. TRABALHOS FUTUROS

Este estudo apresentou um sistema pensado para ser modular e de fácil expansão. Áreas de interesse para trabalhos futuros:

- Treinar um modelo em um *dataset* com mais dados em português.
- Testar modelos diferentes de *embedding*.
- Adicionar à *pipeline* sistemas de instrumentação automatizados que recolham dados e *logs* das conversas para fim de aplicar técnicas de análise do aprendizado.
- Verificar a performance do sistema com um *dataset* maior.
- Implementar um RAG multi-agente onde cada agente possui um nível incremental de conhecimento $n_i = n_{i-1} + 1$. Essa abordagem pode ser útil quando se deseja limitar o conhecimento de um modelo à uma sequência incremental de dados. Por exemplo, cada agente estaria ciente da matéria até certo ponto,

podendo haver a troca de agentes conforme o usuário avança no currículo.

- Adicionar informações sobre o usuário ao sistema como arquivos de provas passadas, trabalhos e notas para personalizar a experiência de modo individual.
- Coletar *feedbacks* dos usuários ao longo de um ou mais semestres para identificar outras limitações não identificadas até agora.

REFERÊNCIAS

- [1] World Economic Forum, "Schools of the future: Defining new models of education for the fourth industrial revolution," 2020. [Online]. Available: https://www3.weforum.org/docs/WEF_Schools_of_the_Future_Report_2019.pdf
- [2] Google for Startups, "Panorama de talentos em tecnologia," 2023. [Online]. Available: https://drive.google.com/file/d/1HdlvsAmLvVHkPW46_7TqTcHnbn7_gB0/view?usp=share_link
- [3] R. M. Hoed, "Análise da evasão em cursos superiores: o caso da evasão em cursos superiores da área de computação," 2017.
- [4] Í. Alvim, R. Bittencourt, and R. Duran, "Evasão nos cursos de graduação em computação no Brasil," 04 2024.
- [5] W. Dávila and L. Giraffa, "Ensino de programação orientada a objetos para iniciantes: Uma metodologia para programação criativa," in *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*. Porto Alegre, RS, Brasil: SBC, 2023, pp. 335–344. [Online]. Available: <https://sol.sbc.org.br/index.php/sbie/article/view/26674>
- [6] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "ChatGPT for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.
- [7] M. Guzdial, *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool Publishers, 2015.
- [8] P. Kinnunen and L. Malmi, "Why students drop out CS1 course?" in *Proceedings of the second international workshop on Computing education research*, 2006, pp. 97–108.
- [9] R. Takács, J. T. Kárász, S. Takács, Z. Horváth, and A. Oláh, "Successful steps in higher education to stop computer science students from attrition," *Interchange*, vol. 53, no. 3, pp. 637–652, Dec 2022. [Online]. Available: <https://doi.org/10.1007/s10780-022-09476-2>
- [10] J. M. Wing, "Computational thinking and thinking about computing," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3717–3725, 2008.
- [11] C. Watson and F. W. Li, "Failure rates in introductory programming revisited," in *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014, pp. 39–44.
- [12] T. Gomes, F. Castro, and P. Tedesco, "Desenvolvendo o pensamento computacional na educação infantil: Um toolkit educacional sobre conceitos de programação baseado em storytelling transmedia," *Sánchez, J*, pp. 31–40, 2017.
- [13] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.
- [14] J. Jeon and S. Lee, "Large language models in education: A focus on the complementary relationship between human teachers and ChatGPT," *Education and Information Technologies*, vol. 28, no. 12, pp. 15 873–15 892, 2023.
- [15] I. Celik, M. Dindar, H. Muukkonen, and S. Järvelä, "The promises and challenges of artificial intelligence for teachers: A systematic review of research," *TechTrends*, vol. 66, no. 4, pp. 616–630, 2022.
- [16] G.-J. Hwang and C.-Y. Chang, "A review of opportunities and challenges of chatbots in education," *Interactive Learning Environments*, vol. 31, no. 7, pp. 4099–4112, 2023. [Online]. Available: <https://doi.org/10.1080/10494820.2021.1952615>
- [17] K. K. de S. OLIVEIRA and R. A. C. de SOUZA, "Digital transformation towards education 4.0," *Informatics in Education*, vol. 21, no. 2, pp. 283–309, 2022.

- [18] C. Hong and W. W. Ma, "Introduction: Education 4.0: Applied degree education and the future of work," in *Applied Degree Education and the Future of Work: Education 4.0*. Springer, 2020, pp. 1–13.
- [19] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.
- [20] Artificial intelligence (AI) vs. machine learning (ML). Google Cloud. [Online]. Available: <https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning>
- [21] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.
- [22] M. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 1969.
- [23] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [24] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017. [Online]. Available: <https://arxiv.org/abs/1710.05941>
- [25] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*. PMLR, 2017, pp. 933–941.
- [26] N. Shazeer, "GLU variants improve transformer," 2020. [Online]. Available: <https://arxiv.org/abs/2002.05202>
- [27] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: <https://arxiv.org/abs/2307.09288>
- [28] Meta AI. (2024, Apr) Introducing Meta Llama 3: The most capable openly available LLM to date. [Online]. Available: <https://ai.meta.com/blog/meta-llama-3/>
- [29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "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, Oct. 2020, pp. 38–45. [Online]. Available: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>
- [31] C. Manning. (2020, Sep) Artificial intelligence definitions. Stanford University Human-Centered Artificial Intelligence. [Online]. Available: <https://hai.stanford.edu/sites/default/files/2020-09/AI-Definitions-HAI.pdf>
- [32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [34] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: <https://arxiv.org/abs/2302.13971>
- [35] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," 2021. [Online]. Available: <https://arxiv.org/abs/2106.09685>
- [36] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," 2020. [Online]. Available: <https://arxiv.org/abs/1811.12808>
- [37] S. M. da Silva Junior, R. A. B. de Freitas, M. A. C. de Moraes, and D. L. V. Costa, "ChatGPT no auxílio da aprendizagem de programação: Um estudo de caso," in *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*. SBC, 2023, pp. 1375–1384.
- [38] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.
- [39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [40] Y. Xu, X. Zhong, A. J. J. Yepes, and J. H. Lau, "Forget me not: Reducing catastrophic forgetting for domain adaptation in reading comprehension," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.
- [41] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.
- [42] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024. [Online]. Available: <https://arxiv.org/abs/2312.10997>
- [43] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.
- [44] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2022. [Online]. Available: <https://arxiv.org/abs/2109.01652>
- [45] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, X. Zhou, E. Wang, and X. Dong, "Better zero-shot reasoning with role-play prompting," 2024. [Online]. Available: <https://arxiv.org/abs/2308.07702>
- [46] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," 2022. [Online]. Available: <https://arxiv.org/abs/2108.07258>
- [47] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," 2023. [Online]. Available: <https://arxiv.org/abs/2303.18223>
- [48] T. Bates, C. Cobo, O. Mariño, and S. Wheeler, "Can artificial intelligence transform higher education?" pp. 1–12, 2020.
- [49] K. K. de Sousa Oliveira, A. da Silva Marcolino, T. P. Falcão, and E. F. Barbosa, "Ensino e aprendizagem de programação na educação 4.0: Um mapeamento sistemático da literatura," *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, pp. 245–255, 2024.
- [50] O. Zawacki-Richter, V. I. Marín, M. Bond, and F. Gouverneur, "Systematic review of research on artificial intelligence applications in higher education—where are the educators?" *International Journal of Educational Technology in Higher Education*, vol. 16, no. 1, pp. 1–27, 2019.
- [51] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [52] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md
- [53] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller *et al.*, "Llama Guard: LLM-based input-output safeguard for Human-AI conversations," 2023. [Online]. Available: <https://arxiv.org/abs/2312.06674>
- [54] E. Beeching, C. Fourier, N. Habib, S. Han, N. Lambert, N. Rajani, O. Sanseviero, L. Tunstall, and T. Wolf, "Open LLM leaderboard," 2023. [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard
- [55] X. Li, T. Zhang, Y. Dubois, R. Taori, I. Gulrajani, C. Guestrin, P. Liang, and T. B. Hashimoto, "AlpacaEval: An Automatic Evaluator of Instruction-following Models," 2023. [Online]. Available: https://github.com/tatsu-lab/alpaca_eval
- [56] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try ARC, the AI2 reasoning challenge," 2018.
- [57] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "HellaSwag: Can a machine really finish your sentence?" 2019.
- [58] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," 2021.
- [59] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WINOGRANDE: an adversarial winograd schema challenge at scale," 2019.

- [60] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," 2021.
- [61] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring How Models Mimic Human Falsehoods," 2022.
- [62] AI@Meta, "LLama 3 - 400B," 2024. [Online]. Available: <https://llama-2.ai/llama-3-400b/>
- [63] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov *et al.*, "Code llama: Open foundation models for code," 2024. [Online]. Available: <https://arxiv.org/abs/2308.12950>
- [64] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [65] P. Zhang, N. Shao, Z. Liu, S. Xiao, H. Qian, Q. Ye, and Z. Dou, "Extending llama-3's context ten-fold overnight," 2024. [Online]. Available: <https://arxiv.org/abs/2404.19553>
- [66] K. Lee, A. Gangidi, and M. Oldham. Building Meta's GenAI Infrastructure. [Online]. Available: <https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/>
- [67] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [68] Ollama, "Ollama." [Online]. Available: <https://github.com/ollama/ollama>
- [69] G. Gerganov, "llama.cpp: Llm inference in c/c++." [Online]. Available: <https://github.com/ggerganov/llama.cpp>
- [70] Brasil, "Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD)," 2018. [Online]. Available: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm
- [71] OpenAccess AI Collective, "Axolotl." [Online]. Available: <https://github.com/OpenAccess-AI-Collective/axolotl>
- [72] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/llama_index
- [73] H. Chase, "LangChain," Oct. 2022. [Online]. Available: <https://github.com/langchain-ai/langchain>
- [74] Z. Nussbaum, J. X. Morris, B. Duderstadt, and A. Mulyar, "Nomic embed: Training a reproducible long context text embedder," 2024.
- [75] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024. [Online]. Available: <https://arxiv.org/abs/2402.03216>
- [76] D. K. J. L. Sean Lee, Aamir Shakir. (2024) Open source strikes bread - new fluffy embeddings model. [Online]. Available: <https://www.mixedbread.ai/blog/mxbai-embed-large-v1>
- [77] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Multilingual E5 text embeddings: A technical report," Tech. Rep., 2024.
- [78] —, "Improving text embeddings with large language models," *arXiv preprint arXiv:2401.00368*, 2023.
- [79] Y.-A. L. Borgne, "OpenAI vs open-source multilingual embedding models," 2024. [Online]. Available: <https://towardsdatascience.com/openai-vs-open-source-multilingual-embedding-models-e5ccb7c90f05>
- [80] "Designing an open-source LLM interface and social platforms for collectively driven LLM evaluation and auditing," OpenWebUI. [Online]. Available: <https://openwebui.com/assets/files/whitepaper.pdf>
- [81] python filesystem spec, "filesystem_spec." [Online]. Available: https://github.com/fsspec/filesystem_spec
- [82] A. Bozkurt, "Tell me your prompts and I will make them true: The alchemy of prompt engineering and generative AI," pp. 111–118, 2024.
- [83] S. M. Bsharat, A. Myrzakhan, and Z. Shen, "Principled instructions are all you need for questioning LLaMA-1/2, GPT-3.5/4," 2024. [Online]. Available: <https://arxiv.org/abs/2312.16171>
- [84] L. S. Lo, "The art and science of prompt engineering: A new literacy in the information age," *Internet Reference Services Quarterly*, vol. 27, no. 4, pp. 203–210, 2023. [Online]. Available: <https://doi.org/10.1080/10875301.2023.2227621>
- [85] —, "The CLEAR path: A framework for enhancing information literacy through prompt engineering," *The Journal of Academic Librarianship*, vol. 49, no. 4, p. 102720, 2023.
- [86] Prompt engineering. OpenAI. [Online]. Available: <https://platform.openai.com/docs/guides/prompt-engineering>
- [87] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Prompting is programming: A query language for large language models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 1946–1969, Jun. 2023. [Online]. Available: <http://dx.doi.org/10.1145/3591300>
- [88] K. Mao, Z. Dou, F. Mo, J. Hou, H. Chen, and H. Qian, "Large language models know your contextual search intent: A prompting framework for conversational search," 2023. [Online]. Available: <https://arxiv.org/abs/2303.06573>
- [89] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [90] What is a Web Application Firewall (WAF)? Cloudflare. [Online]. Available: <https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/>