GPT Teacher: Desenvolvimento de um Agente de LLM para Programação Assistida em Ambiente VSCode

Rhuan Flores Cunha Fernandes

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2025

Rhuan Flores Cunha Fernandes

GPT Teacher: Desenvolvimento de um Agente de LLM para Programação Assistida em Ambiente VSCode

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Diego Nunes Molinos

Monte Carmelo - MG
2025

Este trabalho é dedicado aos meu pais, Anisio Inacio Fernandes e Glayce Camargo Flores da Cunha Fernandes, que sempre estiveram do meu lado dando todo o suporte necessário para a conclusão dessa etapa na minha vida. Agradeço pelo amor, dedicação e confiança que me influenciaram a persistir e vencer este desafio. Sem eles, nada seria possível.

Agradecimentos

A Deus, primeiramente, por me conceder a saúde, a força e a fé necessárias para superar todos os obstáculos e concluir esta importante etapa da minha vida.

A realização desta jornada é um momento não apenas de realização acadêmica, mas de profunda gratidão. Este trabalho é o resultado de um esforço que transcende o individual, construído sobre o apoio, a orientação e o incentivo de muitas pessoas especiais a quem devo meu mais sincero reconhecimento.

À Professora Daniele, que primeiro me guiou neste caminho, minha eterna gratidão. Sua condução segura e precisa foi o alicerce de todo este projeto. Guardo com enorme carinho sua paciência, sabedoria e disponibilidade, que foram fundamentais em cada etapa inicial. Ao Professor Diego, agradeço por ter abraçado este projeto em um momento crucial e por me guiar com maestria até sua conclusão. Sua perspectiva renovada e seu apoio constante na fase final foram indispensáveis para a maturação das ideias aqui apresentadas. Sou grato por sua objetividade, pela segurança que me transmitiu e pela parceria fundamental para lapidar e finalizar esta pesquisa.

Ao corpo docente do curso, expresso minha gratidão por todos os conhecimentos compartilhados, que construíram a base sobre a qual este trabalho foi edificado. Em especial, a alguns mestres que se tornaram mentores e cuja influência levarei por toda a vida. A estes, que com sua paixão e dedicação transcenderam a sala de aula, desafiando meus limites e inspirando um profundo amor pelo conhecimento, carrego uma dívida de gratidão que palavras dificilmente podem expressar.

Aos meus pais, Anisio e Glayce, meus alicerces. Nenhuma palavra seria suficiente para agradecer por todo o amor incondicional, sacrifício e apoio irrestrito. Vocês acreditaram em mim antes mesmo que eu acreditasse, e cada vitória minha é, antes de tudo, de vocês. Obrigado por serem minha força e meu porto seguro.

Por fim, a todos que, direta ou indiretamente, fizeram parte desta história, minha eterna gratidão.

"Toda jornada tem seu último dia. Não tenha pressa."
Noctis Lucis Caelum (Final Fantasy XV))

Resumo

O processo de ensino e aprendizagem em programação de computadores apresenta desafios significativos, que frequentemente resultam em dificuldades de compreensão e desmotivação por parte dos estudantes. Este trabalho apresenta o desenvolvimento e a avaliação do GPT Teacher, um agente de LLM para programação assistida integrado ao ambiente Visual Studio Code (VSCode). O objetivo central foi projetar e implementar uma ferramenta baseada em LLM com foco no suporte ao aprendizado de programação, capaz de apoiar a aquisição de competências em programação de forma mais eficaz do que assistentes genéricos de codificação. A metodologia adotada envolveu a criação de um protótipo funcional baseado em uma arquitetura de agentes duplos: um Agente de Diagnóstico, responsável pela análise técnica do código, e um Agente de Orientação, encarregado de traduzir essa análise em um diálogo construtivo e educativo para o estudante. Os resultados da validação funcional evidenciam que a abordagem proposta é robusta e promissora, confirmando a hipótese de que agentes de LLM, quando estruturados em um sistema especializado, podem atuar como aliados poderosos no processo de ensino-aprendizagem de programação, conciliando rigor técnico e eficácia pedagógica.

Palavras-chave: IA Generativa, LLMs, Engenharia de Prompt, Agentes de LLM, Ensino de Programação.

Lista de ilustrações

Lista de tabelas

Lista de siglas

BPE Codificação por Par de Bytes - *Byte-Pair Encoding*

CoT Cadeia de Pensamento - *Chain of Thought*

GenAI Inteligência Artificial Generativa

IA Inteligência Artificial

LLM Modelos de Linguagem de Grande Escala - *Large Language Models*

NLP Processamento de Linguagem Natural - *Natural Language Processing*

RLHF Aprendizado por Reforço com Feedback Humano - *Reinforcement Learning from Human Feedback*

VsCode *Visual Studio Code*

Sumário

Capítulo 1

Introdução

Conforme Moreira (2024), a crescente digitalização da sociedade impulsionou uma demanda sem precedentes por profissionais de tecnologia, com cursos voltados à área de tecnologia em alta, tornando o ensino de programação uma área de fundamental importância estratégica e econômica. No entanto, o aprendizado de programação é notoriamente desafiador, marcado por uma alta curva de aprendizado, conceitos abstratos e a necessidade de desenvolver um raciocínio lógico rigoroso.

Nesse contexto, diversas abordagens têm sido empregadas para mitigar as barreiras enfrentadas por estudantes iniciantes de cursos de computação, sobretudo na aprendizagem de programação. Entre essas iniciativas, destacam-se a adoção de metodologias ativas de ensino e ambientes interativos de programação, que buscam superar os obstáculos relacionados à abstração conceitual, à motivação e ao desenvolvimento do raciocínio lógico (BERSSANETTE; FRANCISCO, 2021; CALDERON; SILVA; FEITOSA, 2021).

Apesar dos avanços, muitos estudantes ainda encontram dificuldades em consolidar conceitos fundamentais, o que reforça a necessidade de ferramentas que ofereçam suporte contínuo, personalizado e capaz de estimular o raciocínio de programação. É justamente nesse ponto que as tecnologias emergentes de inteligência artificial passam a desempenhar um papel de destaque, ao possibilitar novas formas de apoio ao aprendizado. Nesse cenário, Silva et al. (2024) apresenta um panorama atual da área, destacando o avanço da Inteligência Artificial Generativa (GenAI), em especial dos Modelos de Linguagem de Grande Escala (LLM), como uma tecnologia capaz de remodelar profundamente as práticas pedagógicas.

No contexto do ensino de programação, essas ferramentas assistidas por GenAI, não apenas oferecem suporte automatizado à escrita de código, mas também viabilizam experiências de aprendizagem mais dinâmicas, interativas e personalizadas. Entre as tendências apontadas, destacam-se a utilização de LLMs para fornecer *feedback* imediato sobre erros conceituais e sintáticos, a adaptação das explicações ao nível de conhecimento do estudante e a possibilidade de criar ambientes de aprendizagem mais acessíveis, inclusivos e centrados no aluno (SILVA et al., 2024).

Não obstante, os LLM criam oportunidades para práticas inovadoras, como a geração de exemplos contextualizados, a simulação de diálogos no formato tutor-aluno e o estímulo ao raciocínio crítico por meio de questionamentos guiados. Contudo, este panorama também ressalta desafios relevantes, como a necessidade de calibrar a intervenção pedagógica para evitar respostas excessivamente prontas, o risco de dependência tecnológica e a importância de manter a curadoria docente como elemento central (ZHAI; WIBOWO; LI, 2024). Assim, a literatura converge para o entendimento de que a GenAI, quando aliada a abordagens pedagógicas estruturadas, tem o potencial de atuar como uma ferramenta transformadora no processo de ensino-aprendizagem em programação.

Diante do exposto, este trabalho situa-se na interseção entre a Inteligência Artificial na Educação e a Engenharia de *Software*, com ênfase no desenvolvimento de ferramentas de suporte ao processo de ensino-aprendizagem em programação. Diferentemente dos assistentes de codificação convencionais, cujo foco recai principalmente na automação da escrita de código. A proposta central consiste em adaptar agentes de LLM para atuar como tutores personalizados integrados ao VsCode, ampliando sua função de ambiente de desenvolvimento para também desempenhar o papel de espaço interativo e responsivo de aprendizagem. Com isso, busca-se promover não apenas o apoio técnico, mas, sobretudo, a mediação didática capaz de estimular a compreensão conceitual e o raciocínio crítico dos estudantes.

1.1 Motivação

Conforme Moraes, Costa e Scholz (2022), existem diversas falhas no ensino tradicional de programação, muitas vezes baseado em aulas expositivas e listas de exercícios, que instituem desafios de escalabilidade para oferecer suporte individualizado. Alunos frequentemente se deparam com obstáculos que geram frustração e desengajamento, como a dificuldade em depurar erros lógicos ou em conectar a teoria sintática com a aplicação prática. A necessidade de um *feedback* imediato e personalizado é um dos fatores mais críticos para o sucesso no aprendizado, mas raramente é viável em turmas numerosas.

Ferramentas de assistência de codificação por Inteligência Artificial (IA), representadas por ferramentas como (GitHub Copilot, 2025), revolucionaram a produtividade no desenvolvimento de software. Estes sistemas são proficientes em prever e gerar blocos de código, acelerando o trabalho de programadores, contudo,essas ferramentas são projetadas para auxiliar o processo de codificação e não contribuir no processo de ensino e aprendizagem.

De acordo com Kiesler e Schiffner (2023) e Silva et al. (2024), essas ferramentas, ao oferecer o código correto, gerado instantaneamente, sem um processo cognitivo, instituem a capacidade de se tornarem uma “muleta” que impede o aluno de desenvolver suas próprias habilidades de resolução de problemas.

Com o advento das LLM, surgiram oportunidades para transformar o ensino de programação, ampliando as formas de interação entre estudantes e máquinas (LIU et al., 2025). No entanto, ainda carece de ferramentas que explorem esse potencial sob uma perspectiva genuinamente didática, em vez de priorizar apenas ganhos de produtividade.

Grande parte das soluções atuais se concentra em fornecer respostas prontas à pergunta "qual é a solução?", quando, em um contexto educacional, seria mais relevante fomentar a reflexão: "o que você tentou até agora e por que acha que não funcionou?". Essa lacuna evidencia a necessidade de repensar o papel dos tutores inteligentes, de modo que a aprendizagem crítica e autônoma seja colocada no centro da experiência de programação (FENG; LIU; GHOSAL, 2024). É justamente nesse desafio que se ancora a motivação deste trabalho.

1.2 Problema

O problema central que guia este trabalho é como uma LLM pode ser efetivamente projetada e integrada a um ambiente de desenvolvimento para atuar como uma ferramenta de suporte à aprendizagem de programação, em vez de um mero assistente de autocompletar código, a fim de aprimorar ativamente a aprendizagem e a autonomia de estudantes de programação.

Este problema se desdobra em desafios específicos:

- O Risco da Dependência Passiva: Ferramentas de IA atuais podem fomentar uma abordagem de copiar e colar, na qual o aluno obtém o resultado sem compreender o processo. O desafio é criar um sistema que promova o engajamento ativo e o raciocínio crítico.

- A Lacuna entre Erro e Compreensão: Alunos frequentemente conseguem corrigir um erro de sintaxe, mas falham em compreender a falha lógica subjacente. O desafio é desenvolver um assistente capaz de fornecer explicações conceituais contextuais que conectem o erro a um princípio fundamental da computação.

- A Falta de Suporte Personalizado e Escalável: A tutoria individual é um dos métodos de ensino mais eficazes, mas logisticamente inviável para a maioria das instituições. O desafio é simular essa experiência de tutoria um a um de forma escalável e acessível, diretamente no ambiente de trabalho do aluno.

Buscar resolver as questões aqui levantadas significa não apenas avançar tecnicamente na área de GenAI, mas também oferecer uma contribuição didática relevante para o ensino de programação no ensino superior. O desenvolvimento de um tutor inteligente integrado ao VsCode tem o potencial de ampliar o acesso a uma formação de qualidade, reduzir

barreiras de aprendizagem e preparar de forma mais consistente os futuros profissionais para os desafios complexos do desenvolvimento de *software*.

1.3 Hipótese

Não foram realizados testes com estudantes para propor uma validação científica, desta forma, esta proposta se apoia no termo de hipótese técnica para designar proposições relacionadas à avaliação funcional do artefato de software desenvolvido neste trabalho.

Diante do exposto, assume-se como hipótese que soluções baseadas em agentes complementares e integradas ao VsCode são funcionalmente capazes de identificar problemas em códigos de programação e gerar *feedback* racional que, além de auxiliar o estudante, contribui para reduzir a dependência passiva de ferramentas de GenAI, favorece a compreensão dos problemas enfrentados e oferece suporte personalizado ao processo de aprendizagem do estudante.

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é conceber e implementar um protótipo funcional de agente assistente de inteligência artificial, baseado em um LLM integrado ao VsCode, destinado a atuar como ferramenta de apoio ao desenvolvimento de competências essenciais no processo de aprendizagem de programação.

1.4.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos serão perseguidos:

- ❏ Desenvolver uma arquitetura de software baseada em múltiplos agentes de IA, onde cada agente possui uma especialização pedagógica distinta.
- ❏ Aplicar técnicas avançadas de engenharia de *prompt* para instruir o LLM a atuar como um tutor eficaz.
- ❏ Implementar o protótipo funcional de uma extensão para o VsCode, que servirá como a interface de interação entre o estudante e o sistema de agentes. Esta extensão será responsável por capturar o contexto do código do usuário e apresentar as orientações geradas pela LLM de forma clara e interativa.
- ❏ Validar o protótipo sob a óptica de quantitativo de *tokens* utilizados para geração da instrução e o tempo de inferência para geração da instrução.

Ao alcançar estes objetivos, este trabalho pretende demonstrar a viabilidade do uso de agentes baseados em LLM como suporte integrado ao ensino de programação. Com isso, busca-se gerar conhecimento novo e relevante sobre como a Inteligência Artificial pode ser moldada não apenas como uma ferramenta de produtividade, mas também como uma parceira educacional eficaz na formação de desenvolvedores de *software*.

1.5 Organização da Monografia

Este trabalho está organizado em sete capítulos. A Introdução apresenta o problema, a importância do tema, a hipótese e os objetivos da pesquisa. Em seguida, a Fundamentação Teórica explica os conceitos principais para entender o projeto, como arquitetura Transformer, agentes de IA e os frameworks utilizados, além de apresentar os trabalhos correlatos. O capítulo de Método descreve a abordagem adotada, incluindo a percepção da problemática, as decisões de *design* e de tecnologias. O capítulo de Design e Implementação mostra em detalhes como o GPT Teacher foi construído, apresentando a arquitetura da solução, a modelagem dos agentes e a integração ao VSCode. O capítulo de Experimentação descreve os cenários de teste, as métricas e o procedimento de execução. A Análise de Resultados discute os dados obtidos nos experimentos, avaliando tempo de inferência, consumo de *tokens* e custos. Por último, a Conclusão resume o que foi feito, destaca as contribuições do estudo e sugere os próximos passos.

Capítulo 2

Fundamentação Teórica e Trabalhos Correlatos

Este capítulo apresenta a fundamentação teórica que serve de alicerce para o desenvolvimento deste trabalho. Serão abordados os conceitos essenciais e as tecnologias pertinentes que contextualizam a pesquisa, estabelecendo a base de conhecimento necessária para a compreensão das metodologias e dos resultados que serão discutidos posteriormente.

A exposição parte dos princípios mais elementares, como Tokenização e *Embeddings* Vetoriais, e avança para arquiteturas complexas como o *Transformer*, que é a base dos *Large Language Models*. Em seguida, aborda-se o campo da Inteligência Artificial Generativa (GenAI) com foco na área da educação e o conceito de Agentes de LLM, detalhando técnicas como Engenharia de *Prompt*. Por fim, o capítulo descreve as tecnologias e *frameworks* específicos utilizados, como o LangChain, Agno e a API de extensões do VsCode, conectando a teoria à aplicação prática deste trabalho.

2.1 Processamento de Linguagem Natural

De acordo com Eppright (2021), o Processamento de Linguagem Natural (NLP), ou *Natural Language Processing*, é um campo da inteligência artificial e da ciência da computação que se dedica a capacitar as máquinas a compreender, interpretar, gerar e responder à linguagem humana, seja em formato de texto ou voz. O principal objetivo do NLP é diminuir a barreira de comunicação entre humanos e computadores, permitindo que a interação ocorra de forma mais intuitiva e natural. Para isso, o campo utiliza modelos computacionais para analisar a estrutura e o significado da linguagem, envolvendo desde a análise sintática, que examina a estrutura gramatical das frases, até a análise semântica, que busca extrair o significado e a intenção por trás das palavras.

2.1.1 Tokenização

De acordo com Robison (2025), a tokenização é um processo decisivo no Processamento de Linguagem Natural, que converte texto bruto em unidades menores, os *tokens*, para que possam ser processados por redes neurais.

O conjunto de todos os *tokens* únicos forma o vocabulário dessa rede e a estratégia de tokenização escolhida impacta diretamente sua performance e eficiência dela. As estratégias tradicionais de tokenização apresentam um dilema fundamental, como aponta Robison (2025), a tokenização em nível de palavra, embora intuitiva, é prejudicada por vocabulários massivos e pela sua inabilidade de gerenciar palavras novas.

Em contraste, a tokenização em nível de caractere resolve os problemas de cobertura de vocabulário e flexibilidade, mas acarreta um custo computacional elevado, uma vez que resulta em sequências de entrada significativamente mais longas.

2.1.1.1 Codificação por Par de Bytes

De acordo com Yang et al. (2024), Codificação por Par de Bytes (BPE) é um algoritmo de compressão de dados, projetado para codificar sequências de texto frequentes em códigos menores. Seu propósito foi adaptado para criar um vocabulário de subpalavras, permitindo que termos raros ou desconhecidos sejam representados como uma sequência de unidades menores e conhecidas.

Nesse sentido, Brown et al. (2020) detalha a arquitetura do GPT-3, relatando o pioneirismo da aplicação da tokenização *Byte Pair Encoding*. Essa abordagem, introduzida ainda no GPT-2 e aprimorada no GPT-3, foi fundamental para mitigar o problema de palavras fora do vocabulário.

Em Hugging Face (2025) é explicado o funcionamento do BPE como um processo iterativo que constrói o vocabulário a partir do zero. Inicialmente, o vocabulário é composto por caracteres individuais (ou *bytes*, na técnica mais robusta de *byte-level* BPE, que garante que nenhum caractere seja desconhecido). O algoritmo então analisa o corpus de texto para encontrar o par de *tokens* adjacentes mais frequente e o funde em um novo *tokens*, que é adicionado ao vocabulário.

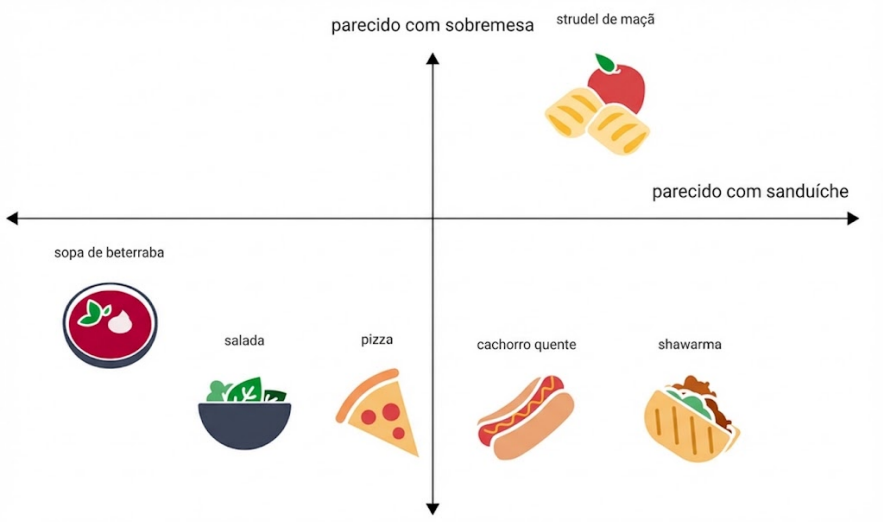
Por exemplo, em um corpus com as palavras “hug”, “pug”, “lug”, “pun” e “gun”, partindo de um vocabulário inicial [“h”, “u”, “g”, “p”, “n”, “l”], o par (“u”, “g”) é o primeiro a ser mesclado, por ser o que mais repete, criando o *token* “ug” e inserindo no vocabulário. Na iteração seguinte, o par mais comum é (“u”, “n”), formando o *token* “un”. Esse ciclo de identificar, fundir e adicionar se repete até que o vocabulário atinja um tamanho predefinido, resultando em um conjunto de *tokens* eficiente que vai de caracteres únicos a palavras inteiras.

2.1.2 *Embeddings* Vetoriais

De acordo com Yao et al. (2025) *embeddings* representam um conceito fundamental em aprendizado de máquina, sendo uma técnica para representar dados de um espaço de alta dimensionalidade em um espaço vetorial de dimensionalidade inferior. O objetivo principal dessa transformação é capturar as relações semânticas e as características dos dados originais em uma representação mais compacta e computacionalmente eficiente.

O Developers (2025) ilustra este conceito com a classificação de alimentos. Inicialmente, propõe-se uma dimensão de "parecido com sanduíche", na qual uma sopa de beterraba teria um valor baixo e um *shawarma*, um valor alto. Essa representação unidimensional, contudo, é insuficiente para classificar um *strudel* de maçã. A solução é adicionar uma segunda dimensão, "parecido de sobremesa", criando um espaço bidimensional onde cada alimento é representado por um vetor de dois elementos. Nesse espaço, o *strudel* teria um valor alto em "parecido de sobremesa" e baixo em "parecido com sanduíche", enquanto um cachorro quente teria o posicionamento inverso. Assim, o exemplo transforma o conceito abstrato de classificação de um alimento em uma representação numérica que captura suas características e relações.

Figura 1 – Alimentos representados em um espaço bidimensional com as dimensões "parecido com sanduíche" e "parecido de sobremesa"



Adaptado de: Developers (2025)

2.1.2.1 Codificação Posicional para Embeddings Vetoriais

Um *Embedding* é uma representação vetorial um ou mais *tokens*, que captura seu significado e contexto semântico. No entanto, quando se processa uma sequência de *tokens*, como um parágrafo, o modelo precisa saber a ordem em que eles aparecem. Em arquiteturas como a *Transformer* Vaswani et al. (2017), que analisam todos os *tokens*

simultaneamente em vez de sequencialmente, a lista de *embeddings* por si só carece de informação sobre a posição de cada *token*.

Para resolver esse problema, o artigo Vaswani et al. (2017) introduz o conceito de Codificação Posicional (*Positional Encoding*). A ideia é injetar um vetor com informações sobre a posição de cada *tokens* na sequência, somando-o diretamente ao seu *embedding*. Os autores propuseram o uso de funções de seno e cosseno com diferentes frequências para calcular essa codificação posicional. As fórmulas são as seguintes:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Onde `pos` é a posição do *token* na sequência, `i` é o índice da dimensão dentro do vetor de *embedding* (de 0 a $d_{\text{model}} - 1$) e $d_{\texttt{model}}$ é a dimensionalidade do vetor de embedding (por exemplo, 512).

Cada dimensão i utiliza uma frequência diferente (controlada pelo termo no denominador), a combinação de senos e cossenos cria um vetor de codificação único para cada posição na sequência. Além disso, essa formulação permite que o modelo aprenda a prestar atenção em posições relativas, uma vez que a codificação para uma posição pos+k pode ser representada como uma função linear da codificação da posição pos.

Dessa forma, para cada *embedding* na sequência, é somado o seu respectivo vetor de *Positional Encoding*, resultando em um vetor final que combina a informação semântica do *token* com a sua posição absoluta e relativa na sequência.

2.2 Inteligência Artificial Generativa

De acordo com a (PAVLIK, 2025), a Inteligência Artificial Generativa, ou simplesmente GenAI, refere-se a um ramo do aprendizado de máquina capaz de criar novos conteúdos, como áudio, imagem e vídeo, a partir de comandos de texto. A interação com esses modelos pode ser notavelmente similar a uma conversa humana, permitindo processos de aprendizado e diálogo por meio de instruções que podem variar em complexidade.

Dessa forma, a GenAI consolida-se como uma vertente inovadora da inteligência artificial, capaz de apoiar processos educacionais ao gerar conteúdos originais a partir da identificação de padrões complexos em grandes volumes de dados. Viabilizada por modelos avançados de *machine learning*, como redes neurais profundas e arquiteturas baseadas em transformadores, essa tecnologia permite a criação autônoma de materiais didáticos, exemplos contextualizados e explicações personalizadas, além de favorecer interações mais dinâmicas e adaptativas entre estudantes e ambientes de aprendizagem digitais.

2.3 Modelos de Linguagem de Larga Escala - LLM

Segundo a Red Hat (2025), um *Large Language Model*, ou simplesmente LLM, trata-se de modelo de inteligência artificial que utiliza técnicas de aprendizado de máquina para compreender e gerar linguagem humana. Assim, segundo a Amazon Web Services [AWS] (2025b), os LLMs são baseados na arquitetura de *Generated Pretrained Transformers* (GPT), uma poderosa estrutura de redes neurais com múltiplas camadas que utiliza mecanismos de codificação e decodificação e auto-atenção.

Conforme descrito na arquitetura do GPT-3 Brown et al. (2020), os LLMs modernos utilizam predominantemente uma arquitetura *decoder-only*, uma especialização da estrutura *Transformer* original. Tona-se importante ressaltar que essa arquitetura é extremamente eficiente na geração de texto, objetivando prever uma distribuição de probabilidades sobre os próximos *tokens* possíveis para completar um contexto.

Para converter as saídas brutas do modelo (os *logits*) nessa distribuição de probabilidade, utiliza-se uma variação da função softmax Kumar (2025). No contexto de geração de texto, essa função incorpora um hiperparâmetro chamado temperatura (T), que permite modular a aleatoriedade e a criatividade das respostas. A função é definida como:

$$\text{Softmax}(z_i, T) = \frac{e^{z_i/T}}{\sum_{j=1}^{K} e^{z_j/T}}$$

Nesta fórmula, z_i é o *logit* associado ao *token* i, K é o número total de *tokens* no vocabulário, e T é a temperatura. Um valor de T mais baixo cria uma distribuição que favorece os *tokens* mais prováveis, gerando respostas mais determinísticas. Por outro lado, um valor mais alto da temperatura, aumenta a probabilidade de *tokens* menos prováveis serem escolhidos, o que resulta em um texto mais variado e criativo.

O desenvolvimento de modelos de larga escala ocorre, de modo geral, em duas etapas fundamentais. A primeira corresponde ao pré-treinamento, no qual o modelo é exposto a um volume massivo de dados textuais extraídos da internet, permitindo-lhe adquirir representações estatísticas da linguagem natural, bem como assimilar conhecimentos factuais e padrões de raciocínio. A segunda etapa, denominada pós-treinamento, visa refinar esse modelo base de forma a torná-lo mais útil e seguro em contextos de interação prática. Esse refinamento é realizado inicialmente por meio do *fine-tuning* supervisionado, que ensina o modelo a seguir instruções e a estruturar respostas com base em exemplos curados de alta qualidade. Em seguida, aplicam-se técnicas como o Aprendizado por Reforço com Feedback Humano (RLHF), responsáveis por alinhar o comportamento do modelo às preferências humanas, de modo a assegurar que suas respostas sejam pertinentes, consistentes e adequadas às expectativas do usuário.

2.3.1 *Transformer*

De acordo com o Ferrer (2024), a arquitetura *transformer* foi originalmente criada com o propósito de aprimorar a tradução automática neural. No entanto, sua aplicação se estende a qualquer problema que envolva a conversão de uma sequência de entrada em uma sequência de saída.

A arquitetura *transformer*, proposta por Vaswani et al. (2017), representou uma mudança de paradigma no Processamento de Linguagem Natural ao solucionar gargalos computacionais inerentes aos modelos sequenciais, como as Redes Neurais Recorrentes, permitindo a paralelização e captura de dependências/relações de longo alcance entre os *tokens* de uma frase. O *transformer* supera esses desafios através de uma arquitetura baseada exclusivamente em múltiplos mecanismos de atenção.

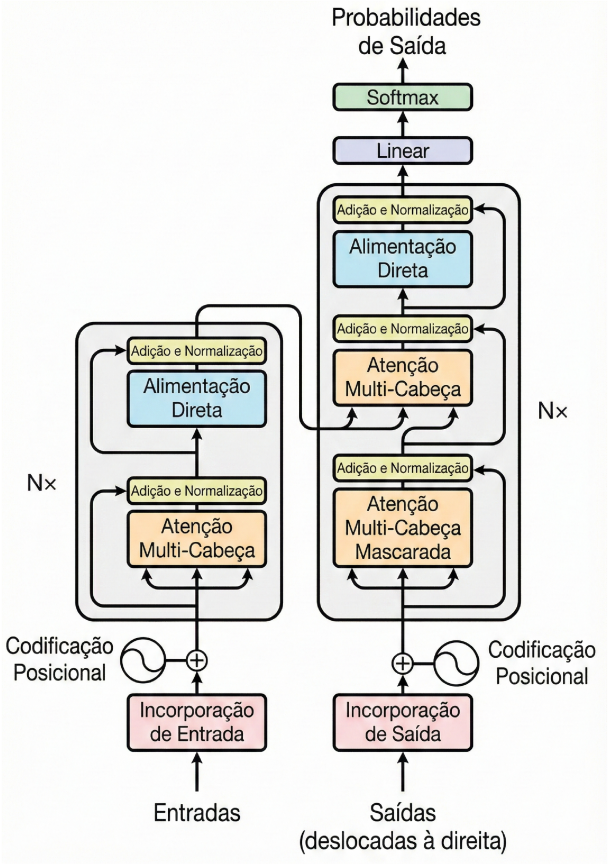
Estruturalmente, o modelo é composto por um empilhamento de codificadores (*Encoders*) e decodificadores (*Decoders*), conforme ilustrado na Figura 2. O codificador mapeia uma sequência de entrada (x_1,...,x_n) para uma sequência de representações contínuas e contextuais $z = (z_1, ..., z_n)$. O decodificador, por sua vez, utiliza z para gerar uma sequência de saída (y_1,...,y_m) de forma autorregressiva. Cada camada, tanto do codificador quanto do decodificador, contém duas subcamadas principais: uma camada de atenção multi-cabeça (*multi-head attention*) e uma rede *feed-forward* conectada. Em torno de cada uma dessas subcamadas, são empregadas conexões residuais seguidas por normalização de camada, técnicas essenciais para estabilizar o treinamento de redes profundas.

O processamento inicia-se com a conversão dos *tokens* de entrada e saída em vetores de representação contínua (*embeddings*). Como a arquitetura baseada em atenção não possui uma noção intrínseca da ordem dos elementos na sequência, é necessário injetar informações posicionais. Para isso, vetores de Codificação Posicional são somados aos *embeddings* dos *tokens*, constituindo a representação final de entrada para a primeira camada do codificador e do decodificador.

No codificador, a subcamada de auto-atenção (*self-attention*) permite que cada *token* da sequência de entrada se relacione com todos os outros, gerando representações enriquecidas pelo contexto global da frase. O resultado da última camada do codificador é um conjunto de vetores que serve de entrada para o decodificador.

O decodificador opera de maneira ligeiramente distinta. Sua primeira subcamada de atenção, denominada auto-atenção mascarada (*masked self-attention*), garante que, ao prever um *token* na posição i, o modelo apenas tenha acesso aos *tokens* anteriores, preservando a propriedade autorregressiva. A segunda subcamada, conhecida como atenção codificador-decodificador (*encoder-decoder attention*), é o ponto central da tradução. Nela, os vetores gerados pela auto-atenção mascarada do decodificador atuam como consultas (*queries*) para recuperar informações da saída do codificador com as respostas (*keys*). Esse mecanismo permite que o decodificador foque nas partes mais relevantes da sequência de entrada para gerar o próximo *token* da sequência de saída.

Figura 2 – Arquitetura de um Transformer



Fonte: Adapatado de Vaswani et al. (2017)

Finalmente, o vetor resultante da atenção codificador-decodificador passa por uma camada linear, que o projeta para a dimensão do vocabulário, seguida por uma função *softmax*. A *softmax*. gera uma distribuição de probabilidade sobre todos os *token* possíveis, indicando qual deles é o mais provável para ser o próximo elemento da sequência, que será incluído à tradução. Este processo é repetido iterativamente até que um *token* de final de sequência seja gerado, concluindo a tradução.

Mecanismo de Auto-Atenção

De acordo com Ferrer (2024) o mecanismo de auto-atenção permite que os modelos relacionem cada palavra na entrada com outras palavras. Por exemplo, em um determinado exemplo, o modelo pode aprender a conectar a palavra *are* (são) com *you* (você).

O artigo Vaswani et al. (2017) explica esta função operando sobre três entradas: um conjunto de vetores de pesos *Query* (Q), *Key* (K) e *Value* (V):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Em um contexto técnico, *Query* (Q), *Key* (K), e *Value* (V) são três representações vetoriais distintas de cada *token* de entrada, geradas a partir da multiplicação do *embedding* do *token* por matrizes de pesos treináveis. A *Query* representa o *token* atual, atuando como uma consulta que busca informações relevantes nos demais *embeddings* da sequência para aprimorar sua própria representação. A *Key* é uma representação de cada *token* na sequência que é comparada (via produto escalar) com a *Query* para calcular um score de alinhamento ou compatibilidade. O *Value* contém a informação semântica do *token* que será efetivamente agregada na saída.

A similaridade entre um vetor *Query* e os vetores *Key* da sequência é calculada através do produto escalar. Matematicamente, isso é expresso pela multiplicação matricial QKT. O resultado é uma matriz de scores, onde cada elemento (i, j) representa a compatibilidade da *Query* do i-ésimo *token* com a *Key* do j-ésimo *token*. Os scores de alinhamento são escalados, sendo divididos pela raiz quadrada da dimensão dos vetores *Key*.

A função *softmax* é aplicada aos scores escalados. Esta operação normaliza os scores, convertendo-os em uma distribuição de probabilidade, resultando no que é conhecido como pesos de atenção. Cada peso, um valor entre 0 e 1, indica a importância relativa de cada *token* na sequência para a representação do *token* atual.

Finalmente, a matriz de pesos de atenção é multiplicada pela matriz de *Value* (V). Esta operação consiste em uma soma ponderada dos vetores *Value*, onde cada vetor é multiplicado pelo seu respectivo peso de atenção. O resultado é um vetor de saída que representa o *token* original, agora enriquecido com informações contextuais de toda a sequência, ponderadas pela relevância de cada *token*.

Para aprimorar este mecanismo, o *Transformer* introduz a atenção multi-cabeça (*Multi-Head Attention*). A auto-atenção é então aplicada em paralelo a cada uma dessas projeções. Os resultados das cabeças são concatenados e novamente projetados linearmente para produzir a saída final. Isso permite que o modelo atenda a informações de diferentes subespaços de representação em diferentes posições simultaneamente, capturando um espectro mais rico de relações contextuais.

Camada de *Feed-Foward*

Outra camada importante é a de *feed-foward*. O artigo Vaswani et al. (2017) define como uma camada que processa o vetor de cada palavra *token* de forma independente, adicionando mais capacidade de aprendizado ao modelo após a camada de atenção ter misturado as informações entre as palavras. Vale ressaltar que cada camada no codificador e no decodificador contém uma rede *feed-foward* e seus pesos são diferentes.

2.4 Agentes de LLM

De acordo com(Amazon Web Services [AWS], 2025a), um agente de inteligência artificial é um *software* que opera de forma autônoma em seu ambiente. Ele coleta informações e as utiliza para realizar tarefas, tomando decisões independentes sobre as ações necessárias para atingir os objetivos que foram previamente definidos por humanos.

Mediante essa perspectiva, um agente de IA pode ser projetado para auxiliar estudantes no aprendizado de programação dentro do ambiente VsCode. Em vez de simplesmente fornecer respostas prontas, esse agente interage com o aluno de forma pedagógica, explicando conceitos fundamentais, sugerindo abordagens e guiando-o passo a passo na resolução do problema. Assim, o agente promove um aprendizado ativo, incentivando o aluno a compreender os conceitos e desenvolver suas próprias soluções.

2.4.1 Engenharia de *Prompt*

Conforme definido por (MESKÓ, 2023), a engenharia de *prompt* é uma área de pesquisa recente focada na criação, aprimoramento e aplicação de instruções (*prompt*). O objetivo dessas instruções é direcionar o que os grandes modelos de linguagem (LLMs) geram, auxiliando-os na execução de diferentes tarefas. Para guiar uma LLM de forma mais precisa e minimizar o risco de alucinações em suas respostas, diversas técnicas podem ser aplicadas. Essas estratégias incluem a definição clara do contexto, a formulação de instruções detalhadas e a padronização do formato das entradas e saídas. Além disso, é possível incorporar exemplos, restrições e regras explícitas para que o modelo siga uma linha de raciocínio específica, promovendo maior confiabilidade e coerência nas respostas.

Damke, Gregorini e Copetti (2024) discute e avalia diversas técnicas de engenharia de *prompt*:

- *Zero Shot*: é uma técnica na qual o modelo realiza uma tarefa sem qualquer exemplo específico fornecido. Dessa forma, o modelo confia apenas no conhecimento obtido durante o treinamento prévio. Essa abordagem é conveniente em casos em que não há exemplos disponíveis ou quando a tarefa é relativamente simples

- *Few-Shot Prompting*: é uma técnica que fornece alguns exemplos para a tarefa desejada antes de solicitar a execução de uma nova instância da tarefa. A utilização dessa técnica é especialmente útil em casos nos quais é necessário que o modelo aprenda padrões personalizados específicos.

- *Chain-of-Thought Prompting*: permite capacidades de raciocínios complexas através de orientar o modelo a analisar o problema em etapas, dividindo a tarefa de problemas a subproblemas ou etapas intermediárias. A abordagem é eficaz em problemas que requerem raciocínio lógico, pois o modelo é capaz de explorar e justificar cada passo do processo de resolução para fornecer uma solução final.

Além disso, Antropic (2025) aconselha dividir tarefas complexas em subtarefas, cada uma executada por um *prompt* ou cadeia de *prompts* diferentes. Essa abordagem aumenta a precisão, a clareza e a rastreabilidade, permitindo a criação de agentes especializados em cada tópico, com tarefas, personas e contextos bem definidos.

2.4.2 Ferramentas

As ferramentas capacitam um agente LLM a interagir com ambientes externos, superando as limitações de seu conhecimento estático. Elas formam um conjunto de interfaces que pode incluir desde APIs de busca, até interpretadores de código, mecanismos de matemática, bancos de dados e modelos externos DAIR.AI (2025). Dessa forma, as ferramentas permitem tanto enriquecer o contexto do agente com informações atualizadas quanto executar rotinas de código para realizar ações específicas quando necessário.

2.5 *Frameworks* de LLM

Frameworks de LLM são conjuntos de ferramentas, bibliotecas e abstrações de software que simplificam e aceleram o desenvolvimento de aplicações complexas construídas sobre LLMs. Em vez de interagir diretamente com a API de um LLM, o que pode ser complexo e repetitivo, um framework oferece componentes reutilizáveis para as tarefas mais comuns. Isso inclui gerenciar e otimizar prompts, conectar o LLM a fontes de dados externas (como documentos ou bancos de dados), encadear múltiplas chamadas ao modelo para realizar tarefas mais sofisticadas e dar ao modelo a capacidade de interagir com outras ferramentas e APIs.

2.6 Langchain

De acordo com Langchain (2025c), o LangChain é um *frameworks* que facilita o desenvolvimento de aplicações baseadas em modelos de linguagem (LLMs). Ele cobre todo o ciclo de vida dessas aplicações: desde a construção com componentes *open-source* e integrações, passando pela monitoração e otimização via LangSmith, até o deployment com suporte para APIs e assistentes usando LangGraph. Além disso, o LangChain oferece uma interface padrão para integração com diversos modelos e ferramentas, como modelos de *embeddings* e bancos vetoriais, tornando-o uma solução completa para criar e escalar soluções baseadas em IA.

2.6.1 *Chains*

No LangChain Langchain (2025a), uma *chain* (ou corrente) é uma sequência de componentes, como modelos de linguagem de diferentes provedores, *prompts* ou outras funções,

interligados de forma a executar uma tarefa mais complexa. A ideia central é que a saída de um componente na sequência serve como entrada para o componente seguinte, permitindo a construção de fluxos de trabalho automatizados e mais elaborados. Isso possibilita que aplicações de inteligência artificial não se limitem a uma única chamada de um modelo de linguagem, mas sim orquestrem múltiplas etapas para, por exemplo, buscar dados em uma fonte, formatá-los com um *prompt*, obter uma resposta do modelo e, em seguida, processar essa resposta.

2.6.2 *Runnables*

De acordo com a documentação do Langchain (2025b), um *runnable* no LangChain é a unidade de trabalho fundamental que compõe as *chains*, representando qualquer componente (como um modelo de linguagem, um *prompt*, ou uma função) que pode ser executado. A principal característica de um *runnable* é que ele implementa uma interface padrão para chamadas de modelos de linguagem, permitindo que diferentes componentes sejam facilmente encadeados com o operador "|", garantindo que eles possam se comunicar de forma previsível e eficiente, facilitando a construção, a reutilização e a depuração de fluxos de trabalho complexos.

2.7 Agno

O Agno (2025) é um *framework* de alta performance projetado para sistemas multi-agente. Ele permite construir, executar e gerenciar sistemas de agentes de forma segura na nuvem. O Agno se destaca por oferecer uma estrutura rápida para o desenvolvimento de agentes, com funcionalidades como gerenciamento de sessão, memória, conhecimento e suporte para interação humana.

2.8 *Visual Studio Code*

Segundo (Microsoft, 2025), o Visual Studio Code é um editor de código-fonte que se destaca por ser leve e, ao mesmo tempo, poderoso. Ele funciona como um aplicativo de desktop e é compatível com os sistemas operacionais Windows, macOS e Linux. Complementando, (Stack Overflow, 2024) relata que o Visual Studio Code é o Ambiente de Desenvolvimento Integrado mais utilizado entre iniciantes na programação. O estudo revelou que 77,6% dos desenvolvedores que estão aprendendo a programar utilizam o VS Code, mais que o dobro da porcentagem alcançada pelo segundo colocado, o IntelliJ IDEA, com 29,9%.

2.8.1 API de Extensões do *Visual Studio Code*

De acordo com a documentação do VS Code (Visual Studio Code, 2025), sua API de Extensão é um conjunto de ferramentas e interfaces de programação, baseada em JavaScript e TypeScript, que funciona como uma ponte para conectar funcionalidades criadas por desenvolvedores diretamente ao núcleo do editor. Na prática, essa API permite a criação de extensões para quase qualquer finalidade, desde adicionar suporte completo a novas linguagens de programação e integrar ferramentas de análise e depuração, até modificar a interface com novos painéis e personalizar totalmente a aparência com temas.

2.9 Trabalhos Correlatos

Nesta seção são apresentados os trabalhos correlatos que dialogam com o tema central desta proposta, o qual se concentra na aplicação da Inteligência Artificial Generativa como suporte ao ensino de programação de computadores. O objetivo é situar a pesquisa no estado da arte, destacando abordagens já exploradas, seus principais resultados e as lacunas que justificam o desenvolvimento do presente estudo.

Em Prather et al. (2023), os autores investigam como programadores novatos utilizam GitHub Copilot (2025), um programador par com IA que oferece sugestões de código inteligentes e conversacionais diretamente no VS Code, para acelerar o desenvolvimento. Através de um estudo de usabilidade com 20 estudantes de graduação com pouca experiência em programação, os pesquisadores observaram como eles resolviam problemas de programação com e sem o auxílio da ferramenta. Os resultados revelaram uma relação dúbia: por um lado, os novatos acharam o *Copilot* extremamente útil para superar bloqueios criativos, gerar código repetitivo e aprender novas sintaxes e bibliotecas, descrevendo a experiência como mágica. Por outro lado, a ferramenta introduziu desafios significativos. A principal dificuldade foi a tendência dos estudantes a uma superconfiança com a ferramenta, aceitando sugestões de código sem compreendê-las completamente, o que levava a longos e frustrantes processos de depuração de erros que eles não sabiam como resolver. Além disso, a constante geração de sugestões era, por vezes, uma fonte de distração, interrompendo o fluxo de raciocínio dos participantes. O estudo conclui que, embora o *Copilot* tenha um grande potencial como ferramenta de andaime, seu design precisava ser aprimorado para incentivar uma compreensão mais profunda do código e mitigar os riscos de dependência e distração para programadores em estágio inicial de aprendizado.

O estudo conduzido por Silva et al. (2025) teve como objetivo realizar um survey com 55 estudantes do curso de Sistemas de Informação da UFSM-FW, analisando o uso de ferramentas de IA generativa, como *ChatGPT* e *GitHub Copilot*, no processo de aprendizagem de programação. Os resultados indicaram que tais ferramentas são amplamente conhecidas e utilizadas, sendo percebidas pela maioria dos participantes como eficazes para esclarecer dúvidas, fornecer exemplos e acelerar a escrita de código. Essa ado-

ção refletiu-se em impactos positivos tanto no desempenho acadêmico (81,9% relataram melhora) quanto na motivação (54,6% perceberam aumento). Entretanto, a pesquisa também evidenciou desafios importantes, como a preocupação com informações incorretas e, sobretudo, o risco de dependência excessiva, admitido por 74,5% dos estudantes, que demonstraram receio quanto ao comprometimento do pensamento crítico e da capacidade de produzir código autoral. Como alternativa para mitigar tais problemas, os alunos sugeriram a implementação de um "modo educacional"nas ferramentas, que privilegie o raciocínio e a construção gradual do conhecimento em vez da simples entrega de respostas prontas. Reforçaram, ainda, a necessidade de um uso consciente e equilibrado, de modo que a tecnologia atue como aliada complementar, e não como substituta do desenvolvimento autônomo das competências de programação.

Em Melo e Moura (2023), os autores defendem que a IA generativa constitui uma poderosa aliada no processo de aprendizagem, atuando como ferramenta para gerar *insights*, estimular a criatividade e aprimorar a resolução de problemas. O estudo apresenta orientações sobre como estudantes e professores podem explorar tais recursos de forma eficaz. Para os alunos, o *ChatGPT* pode funcionar como um tutor personalizado, capaz de corrigir e explicar códigos, realizar conversões entre linguagens, detalhar conceitos e propor exercícios interativos e gamificados. Para os professores, a IA é descrita como um assistente na elaboração de avaliações, na sugestão de projetos, na análise de trabalhos e na oferta de feedback qualificado. O trabalho ainda enfatiza que a chave para o uso adequado está na construção de *prompts* claros e bem definidos, uma vez que a ferramenta possibilita um aprendizado adaptativo e personalizado, ajustando a complexidade e o estilo das respostas às necessidades individuais. Dessa forma, a dinâmica do ensino de programação pode ser significativamente transformada, conciliando apoio pedagógico e inovação tecnológica.

Em Silva et al. (2024), os autores propuseram um estudo com 40 alunos entre 17 e 27 anos e ressaltaram que a geração de código incorreto por inteligência artificial está intrinsecamente ligada a desafios pedagógicos e éticos mais amplos. A confiança cega nas respostas da IA pode levar os alunos a uma dependência excessiva, o que prejudica o desenvolvimento de habilidades cruciais como a resolução de problemas, a depuração de código e o pensamento crítico necessário para validar as soluções. Essa dependência fomenta um uso irresponsável da tecnologia e levanta sérias questões sobre plágio, tornando difícil distinguir o trabalho original do aluno de uma solução gerada artificialmente. Além de comprometer o aprendizado, o uso acrítico da ferramenta arrisca perpetuar vieses presentes nos seus dados de treinamento. Portanto, é essencial uma abordagem pedagógica cuidadosa que ensine os alunos a utilizar essas ferramentas como um complemento, desenvolvendo o senso crítico para questionar e validar as respostas, em vez de substituírem o raciocínio e a validação humana.

Complementando, em Kiesler e Schiffner (2023), os autores reforçam que um dos ris-

cos mais significativos das ferramentas de IA na programação é sua capacidade de gerar códigos que, apesar de parecerem corretos e serem apresentados de maneira convincente, são funcionalmente incorretos. Este fenômeno é particularmente perigoso para estudantes iniciantes, pois a falta de conhecimento aprofundado dificulta a identificação de erros, levando-os a aceitar e internalizar soluções falhas como se fossem corretas. Frequentemente, a origem desses erros não está apenas no modelo de linguagem, mas na própria interação com o usuário: a falta de clareza na pergunta ou uma descrição vaga do problema pode fazer com que a IA interprete a tarefa de forma equivocada, resultando em um código que não resolve a questão pretendida.

2.9.1 Síntese dos Trabalhos Correlatos

Os achados do mapeamento revelam o duplo papel da GenAI no ensino de programação. Por um lado, as ferramentas demonstram grande potencial pedagógico, ao possibilitar a resolução de exercícios, a explicação de códigos complexos, a oferta de *feedback* imediato e a correção de erros, fatores que contribuem para aumentar a autoeficácia, a motivação e o engajamento dos estudantes. Além disso, estudos indicam que tais recursos podem atuar como tutores personalizados, capazes de adaptar explicações ao nível de conhecimento do aprendiz e fornecer exemplos contextualizados que favorecem a compreensão.

Por outro lado, os trabalhos também alertam para riscos relevantes, como a confiança excessiva nas respostas da IA, a dependência passiva na geração de soluções prontas, a dificuldade em identificar erros quando o código parece convincente, além de questões éticas associadas ao plágio e à perpetuação de vieses presentes nos dados de treinamento. Tais fatores podem comprometer o desenvolvimento de competências fundamentais, como o raciocínio lógico, a depuração autônoma de código, a validação crítica das soluções e a autoria intelectual do estudante.

Nesse sentido, a literatura converge para a necessidade de que a integração da GenAI em contextos educacionais seja orientada por estratégias pedagógicas estruturadas, que promovam não apenas a produtividade técnica, mas também a formação crítica e autônoma do aprendiz. Essa lacuna abre espaço para o desenvolvimento de soluções que conciliem rigor técnico e mediação didática, como a proposta deste trabalho, que busca estruturar a interação entre IA e estudante de forma a estimular o raciocínio crítico e reduzir o risco de dependência passiva.

Tabela 1 – Síntese dos trabalhos correlatos e sua relação com a proposta

Trabalho	**Foco da Pesquisa**	**Principais Achados e Desafios**	**Relação com o GPT Teacher (Proposta)**
Prather et al. (2023)	Usabilidade do GitHub Copilot por iniciantes.	Auxilia em bloqueios criativos e na geração de código, mas gera superconfiança e falta de compreensão.	Justifica o foco em fornecer explicações em vez de apenas código pronto.
Silva et al. (2025)	Percepção de alunos de SI sobre o uso de Inteligência Artificial.	Melhora o desempenho, porém 74,5% admitem risco de dependência excessiva.	Valida a demanda por um "modo educacional"que instigue o raciocínio.
Melo e Moura (2023)	IA como tutor personalizado e assistente docente.	Potencial para correção, conversão de linguagens e exercícios gamificados.	Serve de base para a definição da persona e do tom de voz pedagógico do agente.
Silva et al. (2024)	Desafios pedagógicos e éticos no uso da IA.	A confiança cega prejudica o pensamento crítico e a autoria do código.	Reforça a necessidade de mediação didática e do conceito de "andaime pedagógico".
Kiesler e Schiffner (2023)	Avaliação de modelos de linguagem para resolver problemas programação.	Os modelos acertam grande parte dos problemas, entretanto, suas alucinações entregam códigos aparentemente corretos e de maneira convincente, dificultando a detecção de erros por estudantes iniciantes.	Apoia a arquitetura de agentes especializados para garantir maior precisão técnica.

Fonte: Autoria Própria

Capítulo 3

Método

Este capítulo descreve o percurso metodológico e técnico adotado para a concepção e implementação do sistema de Ensino Assistido por Inteligência Artificial, com foco na criação de agentes de inteligência artificial generativa voltados à orientação pedagógica integrada ao ambiente de desenvolvimento VsCode. Serão detalhadas, a seguir, as etapas essenciais do projeto, incluindo a definição da arquitetura do sistema, a aplicação de técnicas de engenharia de prompt para moldar as interações pedagógicas e o desenvolvimento da ferramenta sob a forma de extensão funcional no VsCode. O propósito é apresentar, de maneira clara e sistemática, as decisões de design, as tecnologias empregadas e os desafios enfrentados, evidenciando como tais elementos se articularam para a materialização de uma ferramenta capaz de oferecer suporte contextualizado, responsivo e inteligente a estudantes e desenvolvedores em seu ambiente de codificação.

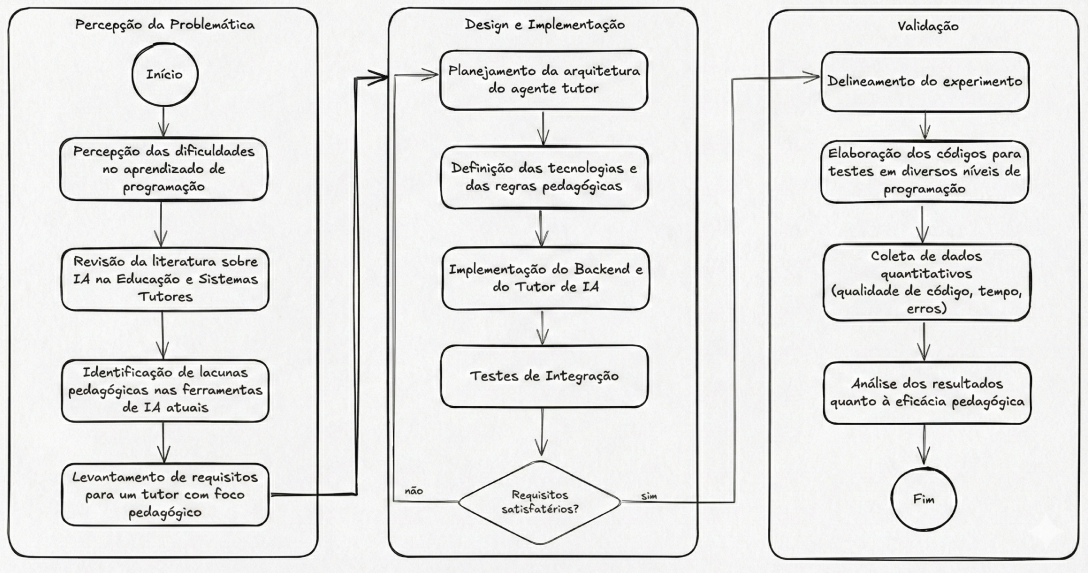
Com o intuito de oferecer uma visão geral do método adotado nesta pesquisa, a Figura 3 apresenta, de forma esquemática, as principais etapas que compõem o percurso metodológico. O diagrama sintetiza desde a concepção inicial do problema até o desenvolvimento técnico do protótipo, abrangendo a definição da arquitetura do sistema, a aplicação da engenharia de prompt e a implementação da extensão no VsCode.

3.0.1 Percepção da Problemática

A etapa de Percepção da Problemática constituiu o ponto de partida deste trabalho e teve como finalidade compreender os desafios inerentes ao processo de ensino-aprendizagem de programação no ensino superior. Inicialmente, foram identificadas, através da literatura, as principais dificuldades enfrentadas por estudantes em estágios iniciais, tais como a elevada carga cognitiva associada à abstração de conceitos, a necessidade de desenvolver raciocínio lógico estruturado e a tendência à desmotivação diante de erros recorrentes.

Na sequência, realizou-se uma revisão da literatura sobre Inteligência Artificial aplicada à Educação e sistemas tutores, com o objetivo de mapear o estado da arte e compre-

Figura 3 – Fluxograma do método



Fonte: do Autor

ender de que forma tecnologias emergentes, como a Inteligência Artificial Generativa, vêm sendo utilizadas nesse contexto. Essa análise permitiu identificar tanto avanços relevantes quanto limitações persistentes nas soluções existentes.

Dentre essas limitações, destacou-se a carência de ferramentas que priorizem a mediação pedagógica em detrimento da simples automação da escrita de código. A maioria dos assistentes atuais responde à lógica da produtividade, oferecendo soluções prontas que, embora acelerem a codificação, podem induzir a uma postura passiva do estudante, reduzindo o engajamento crítico e comprometendo a construção de autonomia intelectual.

Com base nesse diagnóstico, procedeu-se ao levantamento de requisitos para a concepção de um tutor de IA, capaz de oferecer suporte personalizado, promover a reflexão sobre os erros cometidos e estimular o desenvolvimento gradual das competências de programação. Essa etapa, portanto, forneceu o alicerce conceitual e prático para o design do protótipo proposto.

3.0.2 Design e Implementação

As etapas de Design e Implementação corresponderam à fase em que se materializaram as decisões derivadas do levantamento de requisitos pedagógicos e técnicos. Inicialmente, foi elaborado o planejamento da arquitetura do agente tutor, estabelecendo-se a divisão entre os componentes de diagnóstico e de orientação pedagógica, de modo a garantir a separação das responsabilidades técnicas e didáticas.

Em seguida, foram definidos as tecnologias de suporte e o apoio didático, assegurando que as interações com o estudante fossem estruturadas de maneira construtiva, evitando respostas prontas e estimulando o raciocínio crítico. A partir dessas definições, procedeu-

se à implementação do *backend* e do agente de IA, que integraram tanto os mecanismos de processamento do código quanto a camada de diálogo pedagógico.

Posteriormente, foram conduzidos testes de integração para verificar a consistência do sistema como um todo, avaliando a comunicação entre módulos, a adequação das respostas geradas e o alinhamento às diretrizes pedagógicas estabelecidas. Sempre que identificadas falhas ou inconsistências, ajustes foram realizados até que os requisitos definidos fossem plenamente atendidos.

3.0.3 *Stack* de Tecnologias e Ferramentas

A seleção das tecnologias para este projeto foi pautada pela busca de um ecossistema coeso e de alta performance, capaz de suportar as demandas de um sistema de IA e, ao mesmo tempo, integrar-se nativamente ao ambiente de desenvolvimento. A arquitetura foi dividida em dois componentes principais: o *backend*, responsável pela lógica da API e interação com a IA, e a extensão da IDE, que serve como interface para o usuário.

Tecnologias do *Backend*

- Python[1]: Foi a linguagem de programação escolhida como base para o *backend* devido ao seu vasto ecossistema de bibliotecas para Inteligência Artificial, manipulação de dados e desenvolvimento web. Ele permite uma prototipagem rápida e facilita a construção de sistemas complexos de IA devido suas diversas bibliotecas.

- FastAPI: Para a construção da API, foi utilizado o [2]https://fastapi.tiangolo.comFastAPI, um micro-framework web moderno e de alta performance para Python. Sua escolha se deve à sua velocidade, à geração automática de documentação interativa (via OpenAPI[3]) e ao seu sistema de injeção de dependências, que simplifica a organização do código. O FastAPI foi fundamental para a criação de *endpoints* robustos e eficientes para as rotas da aplicação.

- LangChain: No início do projeto, o framework foi empregado para agilizar a manipulação e o encadeamento de prompts com o Modelo de Linguagem. Contudo, em fases posteriores e visando otimizar a latência ele foi retirado.

- Agno: O Agno é um framework para criação e orquestração de agentes de IA que busca uma menor latência para sistemas multi-agênticos.

- SQLModel: Para o mapeamento objeto-relacional (ORM), a biblioteca SQLModel[4] foi utilizada. Ela combina o poder do SQLAlchemy com a validação de dados

[1] https://www.python.org
[2]
[3] https://swagger.io/specification
[4] https://sqlmodel.tiangolo.com

do Pydantic[5], integrando-se perfeitamente ao FastAPI. O SQLModel simplificou a definição dos modelos de dados, a interação com o banco de dados e garantiu a consistência e a validação das informações manipuladas pela API.

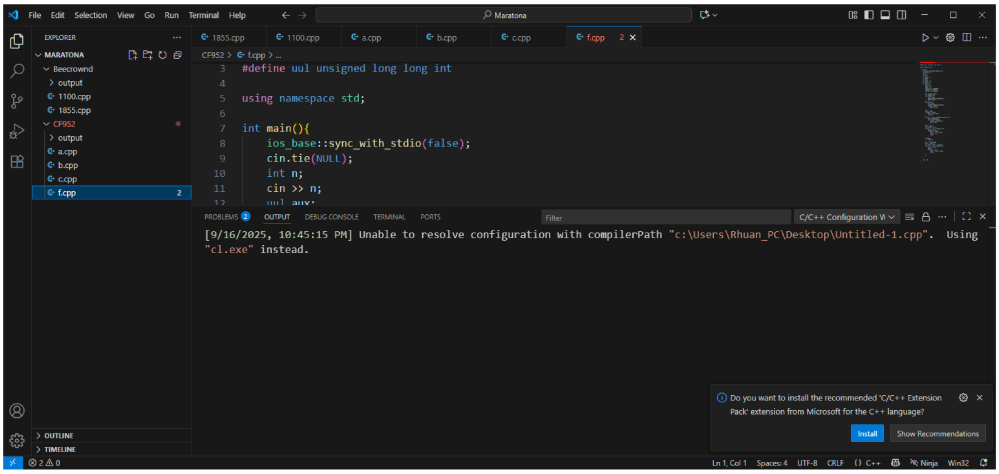
- ❏ PostgreSQL: Como sistema de gerenciamento de banco de dados, o PostgreSQL[6] foi selecionado por sua robustez, confiabilidade e suporte a tipos de dados complexos. Sua capacidade de lidar com cargas de trabalho concorrentes e sua extensibilidade o tornaram a escolha ideal para persistir os dados da aplicação de forma segura e escalável.

Tecnologias da Extensão (Frontend):

- ❏ Node.js e TypeScript: A extensão para o VSCode foi desenvolvida sobre a plataforma Node.js[7], que é o ambiente padrão para a execução de extensões, em conjunto com o TypeScript[8].

- ❏ API de Extensões do Visual Studio Code: A integração com o editor foi realizada por meio da API oficial de extensões do Visual Studio Code (2025). Essa bilbioteca permite a criação de componentes na IDE de usuário, o registro de comandos na paleta do editor e o acesso ao contexto do código do usuário, viabilizando a experiência do tutor integrado à ferramenta. Podemos ver a interface da ferramenta na Figura 4

Figura 4 – Interface principal do editor de código Visual Studio Code



Fonte: do Autor

[5] https://docs.pydantic.dev/latest
[6] https://www.postgresql.org
[7] https://nodejs.org/pt
[8] https://www.typescriptlang.org

3.0.4 Validação

Para este trabalho, a etapa de Validação compreendeu verificar a funcionalidade do protótipo em relação às métricas funcionais e de desempenho da proposta. Para isso, foram avaliados principalmente o tempo de inferência, tanto do agente isoladamente quanto considerando a carga de trabalho, e a quantidade de *tokens* necessárias para materialização dos prompts gerados durante as interações.

Esses testes permitiram mensurar a eficiência do sistema em termos de desempenho computacional e consumo de recursos, assegurando que a ferramenta pudesse operar de forma responsiva e em tempo hábil dentro do ambiente de desenvolvimento VsCode.

Ainda que esta etapa não tenha contemplado experimentos com estudantes, a análise funcional forneceu indícios relevantes sobre a robustez do protótipo e sua adequação para suportar cenários reais de uso, constituindo-se como um passo inicial para futuras investigações voltadas à eficácia educacional do agente tutor.

Posteriormente, foi realizada a análise dos resultados, considerando não apenas os aspectos técnicos da ferramenta, mas também sua capacidade de oferecer suporte didático significativo. Essa avaliação possibilitou identificar pontos fortes do protótipo, bem como limitações a serem superadas em trabalhos futuros.

Assim, a fase de validação cumpriu o papel de consolidar a viabilidade do sistema proposto, fornecendo evidências de que a arquitetura desenvolvida é funcional e que a abordagem baseada em agentes complementares pode contribuir de forma efetiva para o processo de ensino-aprendizagem de programação.

Capítulo 4

Design e Implementação do GPT Teacher

Esta seção apresenta o percurso de design e implementação do protótipo GPT Teacher, cujo objetivo foi estruturar uma arquitetura de agentes de inteligência artificial generativa com foco didático no processo de aprendizagem de programação, integrada ao ambiente de desenvolvimento VsCode. O processo contemplou desde o planejamento conceitual da arquitetura até a implementação efetiva da extensão funcional, buscando garantir tanto a viabilidade técnica quanto a adequação da solução.

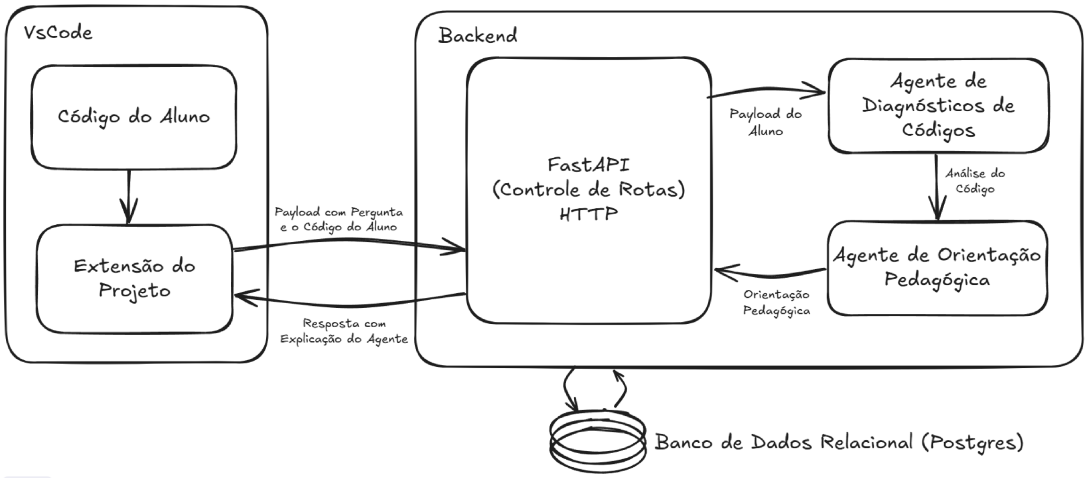
4.1 Arquitetura Geral da Solução

A arquitetura do sistema foi projetada seguindo um modelo cliente-servidor Kurose e Ross (2013), visando o desacoplamento entre a interface do usuário, no ambiente VsCode, e o processamento de inteligência artificial, localizado no *backend*. Essa separação de responsabilidades, ilustrada na Figura 5, garante modularidade, escalabilidade e a centralização da lógica de negócio. A solução é composta por três camadas principais: o cliente (Extensão do Projeto), o servidor (*Backend*) e a camada de persistência (Banco de Dados).

O fluxo de operação se inicia no ambiente do cliente, o VsCode. A Extensão do Projeto atua como a interface para o usuário, permitindo que ele, a partir de seu código no ambiente, submeta uma pergunta ou solicite uma análise. A extensão é responsável por capturar o trecho de código ativo no editor e a dúvida do aluno e encapsulá-los em um *payload*. Este é então enviado via uma requisição HTTP para o *backend*. A imagem 6 mostra um exemplo onde o usuário faz uma pergunta para o agente com o contexto do código ativo da IDE e a figura 7 o corpo da mensagem enviado ao servidor.

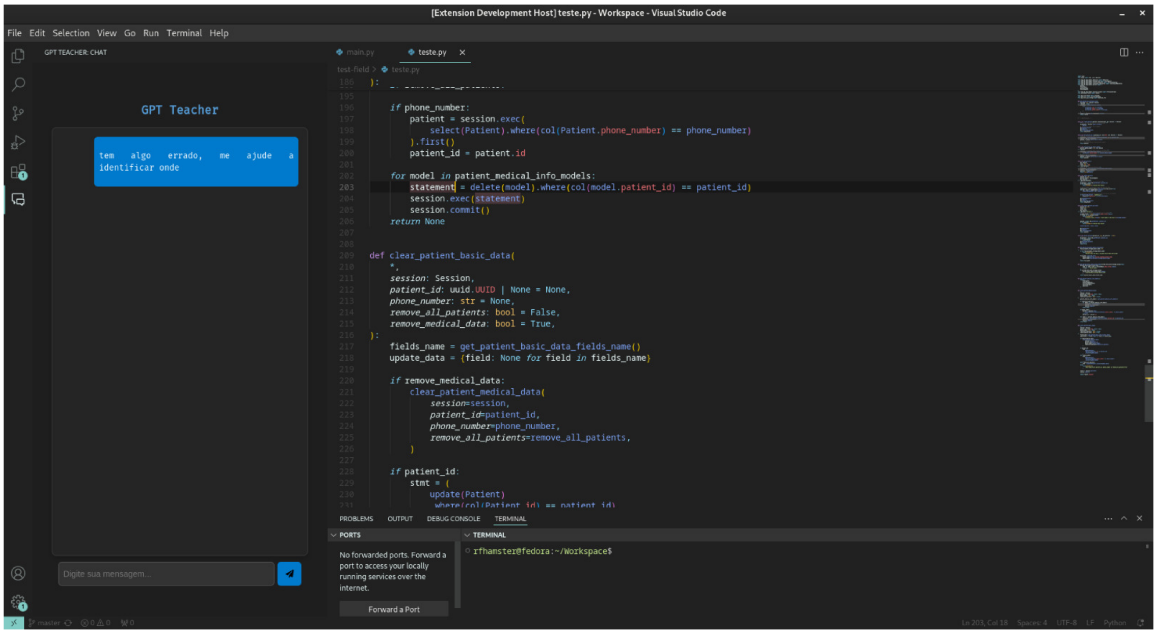
No servidor, a aplicação FastAPI funciona como um API Gateway, controlando as rotas e orquestrando o fluxo de trabalho interno. Ao receber a requisição da extensão, o FastAPI direciona o *payload* do aluno para o Agente de Diagnósticos de Códigos. Este

Figura 5 – Arquitetura Geral da Solução



Fonte: do Autor

Figura 6 – Extensão do VsCode com Perguntas do Aluno para o Agente



Fonte: do Autor

Figura 7 – Corpo da Mensagem Enviado ao Servidor - Fonte: do Autor


```
{
  "message":"tem algo errado, me ajude a identificar onde",
  "code":"import uuid\nfrom typing import Any, List, Optional\n\nfrom angi_db.angi.models.addiction impo
}
```


Fonte: do Autor

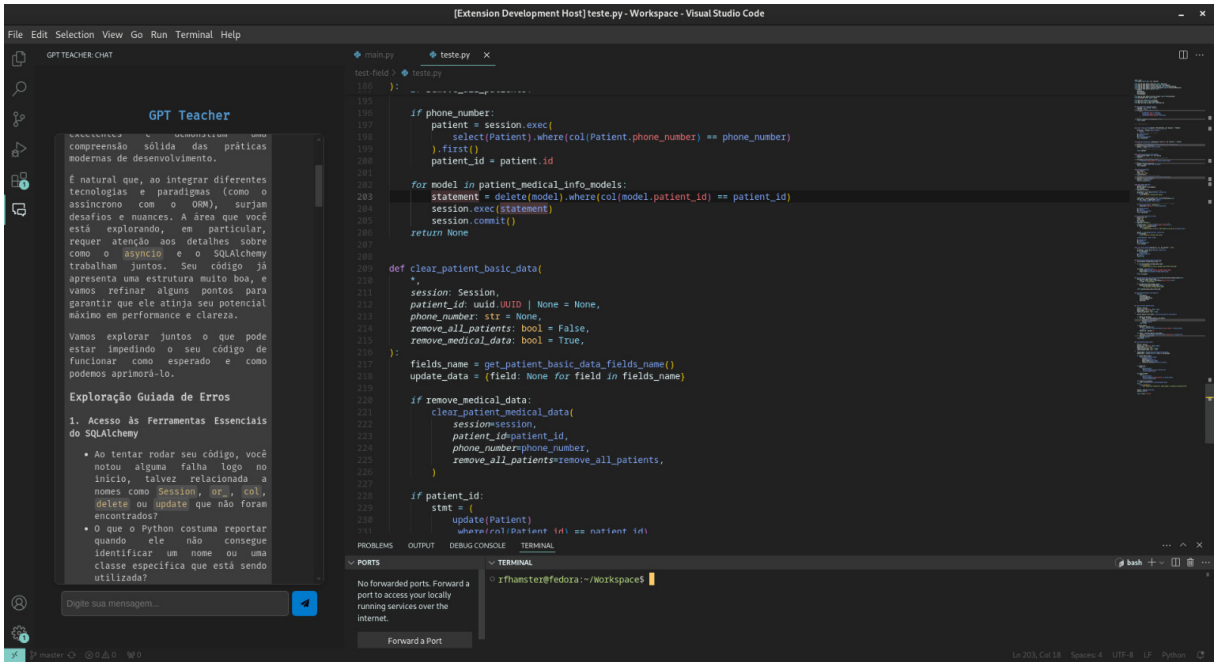
agente especializado tem a única responsabilidade de realizar uma análise técnica e objetiva do código recebido, identificando erros de sintaxe, problemas de lógica, desvios de

boas práticas ou outras anomalias. O resultado deste processo é uma Análise do Código estruturada, exemplificada no anexo A.1.3, que serve de insumo para a próxima etapa.

A análise técnica é então encaminhada ao Agente Tutor. Este segundo agente atua como o tutor do sistema, sua função é traduzir o diagnóstico técnico em um diálogo construtivo claro, didático e eficaz. Em vez de simplesmente apontar o erro, este agente é projetado para guiar o aluno em seu raciocínio, fazer perguntas socráticas e explicar os conceitos subjacentes, promovendo um aprendizado efetivo.

Finalmente, a orientação gerada é formatada pelo FastAPI, que responde à requisição da Extensão do Projeto no VsCode. A extensão, por sua vez, exibe essa resposta de forma clara e integrada à interface do editor, como pode ser visto na imagem 8. Todo o processo é suportado por um Banco de Dados Relacional, que persiste o histórico de interações, gerencia dados de usuários e armazena as saídas geradas pelos agentes.

Figura 8 – Extensão do VsCode com a Resposta Produzida pelo Agente Tutor



Fonte: do Autor

4.1.1 Seleção da Plataforma de Desenvolvimento e Integração

A escolha do Visual Studio Code como ambiente para a aplicação da ferramenta foi uma decisão estratégica. Em vez de desenvolver um software independente, que poderia introduzir uma barreira para utilização, foi criada uma solução integrada a uma ferramenta de uso diário por estudantes. O VsCode se destaca por sua alta popularidade e influência no ecossistema de tecnologia, sendo frequentemente a primeira porta de entrada para programadores em diversas áreas, como desenvolvimento *web*, C++ e outras.

A experiência de uso (UX) no VsCode também foi aprimorada ao longo do projeto. Inicialmente, a ferramenta era ativada por um atalho de teclado. Para melhorar a usabilidade e a visualização dos resultados, essa abordagem evoluiu para uma *View* customizada, acessível por um clique na barra de atividades do editor. Dessa forma, a ferramenta abre uma nova aba com uma interface *web* (*WebView*), tornando o acesso mais intuitivo e a apresentação dos *feedbacks* pedagógicos mais rica e organizada.

4.1.2 Modelagem dos Agentes de IA

A solução foi arquitetada em torno de um sistema multiagente, composto por duas entidades de LLM especializadas e complementares: um Agente Analisador de Código e um Agente Tutor Pedagógico. Ambos foram construídos utilizando uma abordagem de *prompt engineering* baseada na técnica de Cadeia de Pensamento (CoT), que instrui o modelo a pensar passo a passo antes de gerar uma resposta final, aumentando a coerência e a precisão.

O primeiro agente foi projetado para realizar uma análise técnica aprofundada do código submetido pelo aluno. Para garantir a qualidade e o rigor da avaliação, o prompt contextualiza o LLM para atuar sob a persona de um desenvolvedor sênior com mais de dez anos de experiência.

O segundo agente recebe como entrada a análise estruturada gerada pelo primeiro. Sua função é traduzir o diagnóstico técnico em um diálogo construtivo eficaz e personalizado. Para isso, foi configurado com a persona de um Professor de Programação Experiente, especialista em pedagogia e no método socrático.

É importante ressaltar que esta arquitetura final é fruto de um processo de desenvolvimento iterativo. A primeira versão do sistema utilizava um único agente monolítico, responsável tanto pela análise técnica quanto pela tutoria. Contudo, observou-se que essa abordagem limitava a profundidade em ambas as áreas. A decisão de dividir as responsabilidades em dois agentes distintos permitiu um refinamento muito mais focado, onde os prompts de cada um puderam ser aprimorados e otimizados de forma independente ao longo de vários ciclos para alcançar uma maior qualidade nas respostas.

4.1.2.1 Seleção do *Framework* para Construção dos Agentes de IA

O processo de seleção das ferramentas de desenvolvimento dos agentes passou por um ciclo de iteração e refinamento. Inicialmente, o protótipo foi desenvolvido utilizando o framework [1]LangChain. Embora seja uma ferramenta poderosa para a construção de aplicações com LLMs, foram identificados gargalos de desempenho, resultando em uma latência de resposta maior que a desejada. Além disso, a orquestração da comunicação entre os agentes poderia ser um desafio técnico para o futuro.

[1] https://www.langchain.com

Em busca de maior performance, escalabilidade e uma arquitetura mais preparada para evoluções futuras, o projeto foi refatorado utilizando o framework [2]Agno. Esta escolha se mostrou mais eficiente, proporcionando maior velocidade de processamento e integrações que favoreceram a arquitetura planejada.

4.2 Construção dos Agentes

O núcleo inteligente da ferramenta é materializado por meio de uma arquitetura de dois agentes especializados que operam de forma complementar. O primeiro agente é responsável por realizar uma análise técnica e objetiva do código-fonte, enquanto o segundo se encarrega de sintetizar essa análise e construir uma interação didática com o usuário. As subseções a seguir detalham a concepção, as responsabilidades e a implementação de cada um desses componentes.

4.2.1 Agente de Diagnóstico de Código

O Agente de Diagnóstico de Código é o primeiro componente do núcleo de inteligência do sistema, atuando como o perito técnico responsável por realizar uma análise profunda e objetiva do código submetido pelo aluno. Sua principal função é decompor o código em seus elementos fundamentais, identificar padrões, classificar o nível de proficiência demonstrado e categorizar erros.

Este agente não interage diretamente com o aluno e seu comportamento do agente é inteiramente governado por um processo de engenharia de prompt. Este não é um simples comando, mas um [3]*prompt* complexo que define uma persona, um processo rigoroso e um formato de saída estrito. A persona designada ao LLM é a de um Analista de Programação Sênior, estabelecendo um contexto de mentoria e avaliação construtiva que guia todo o raciocínio do modelo.

Um dos principais desafios no trabalho com LLMs é garantir a consistência e a confiabilidade da saída. Para mitigar esse problema, foi implementada uma saída estruturada em formato JSON, técnica que obriga o agente a preencher campos específicos e, assim, força um processo de raciocínio sistemático. Este processo se inicia com uma Análise Inicial (Reconhecimento) para identificar a linguagem, o objetivo do código e os tópicos envolvidos, seguida pela Avaliação de Nível (Classificação) da proficiência do aluno. Subsequentemente, é realizado um Diagnóstico de Erros detalhado, que categoriza as falhas encontradas (sintáticas, lógicas, estruturais, de performance e de boas práticas), culminando na geração de Feedback e Melhorias, com retornos construtivos e sugestões de código.

[2] https://github.com/agno-agi
[3] Prompt presente no anexo A.1.1

Figura 9 – Modelo de Saída Estruturada do Agente de Diagnóstico de Código

Fonte: do Autor

A estrutura de saída, conforme detalhado na imagem 9, foi fundamental para que cada campo servisse de contexto para o próximo, criando uma cadeia de análise lógica e detalhada. Além da rigidez estrutural de sua saída, o agente também se destaca por sua flexibilidade: ele foi projetado para ser altamente configurável, com parâmetros como nivel_detalhamento e tom_feedback que podem ser ajustados dinamicamente. Essa capacidade permite que o sistema adapte a profundidade e o estilo da análise técnica conforme o contexto da interação ou as preferências do usuário. Por fim, a saída estruturada do agente organizada em seções claras permite que o Agente Tutor consuma a informação de forma programática.

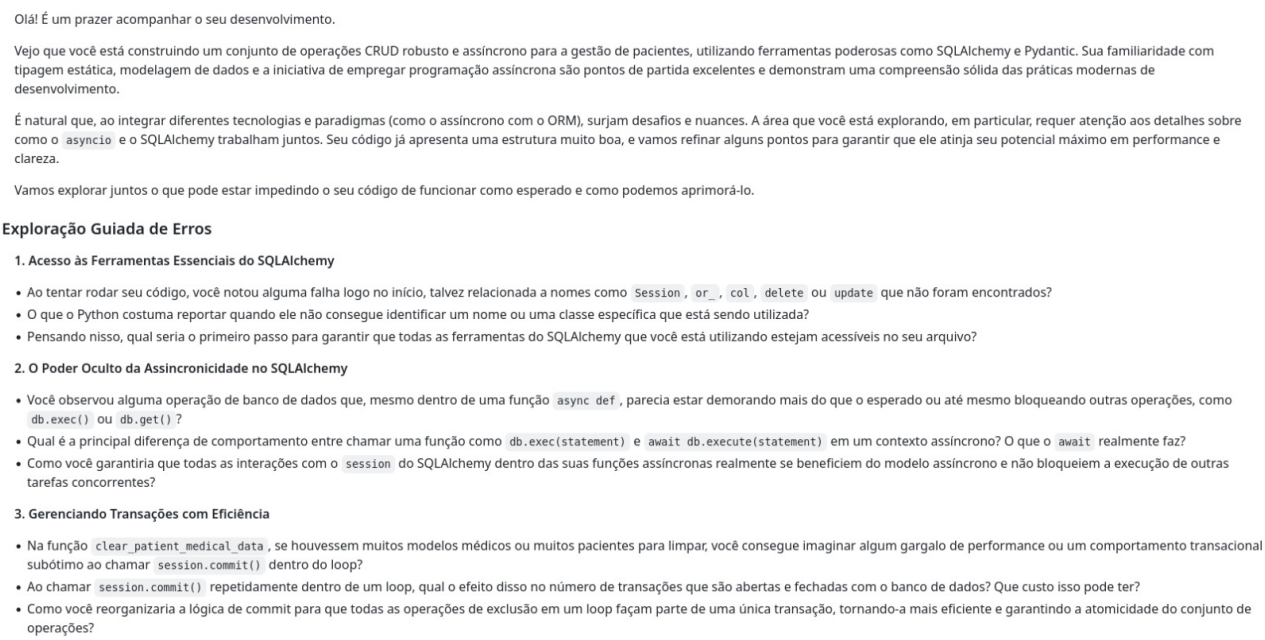
4.2.2 Agente Tutor

O Agente de Tutor conversa diretamente com o aluno, funcionando como o tutor do sistema. Ele não analisa o código, mas recebe um relatório técnico detalhado do Agente de Diagnóstico. A sua principal função é manipular esse relatório técnico e transformá-lo em uma conversa educativa, que ajude o aluno a aprender e a pensar por conta própria.

Todo o comportamento do agente é controlado por um [4]*prompt* muito detalhado que o instrui a agir como um professor especialista em pedagogia aplicada. A sua diretriz principal é nunca dar respostas prontas e maximizar o desenvolvimento do raciocínio autônomo do aluno. Para alcançar este objetivo, o agente adota o método socrático para fazer com que o próprio aluno chegue à solução, estimulando o pensamento crítico, desenvolvendo o raciocínio do estudante e não apenas corrigindo o erro.

Antes de responder, o agente primeiro faz uma análise do aluno. Ele julga o nível de conhecimento do estudante fornecido na análise do agente anterior. Com base nisso, ele decide a melhor forma de explicar os conceitos, usando uma linguagem mais simples para iniciantes ou discussões mais complexas para os avançados. Isso é exemplificado na imagem 10, onde o contexto é um aluno avançado, então o agente explica conceitos como assincronicidade do SQLAlchey/SQLModel, em contraste ao contexto iniciante da resposta presente na 11, onde até o tratamento inicial e a abordagem são diferentes. Além disso, ele prioriza os erros de maior impacto para o aprendizado e identifica possíveis lacunas conceituais, o que garante os pontos mais importantes para o aprendizado naquele momento.

Figura 10 – Exemplo de Resposta à um Aluno nível Avançado referente uma Possível Melhora de Padrão em Operações do SQLModel



Olá! É um prazer acompanhar o seu desenvolvimento.

Vejo que você está construindo um conjunto de operações CRUD robusto e assíncrono para a gestão de pacientes, utilizando ferramentas poderosas como SQLAlchemy e Pydantic. Sua familiaridade com tipagem estática, modelagem de dados e a iniciativa de empregar programação assíncrona são pontos de partida excelentes e demonstram uma compreensão sólida das práticas modernas de desenvolvimento.

É natural que, ao integrar diferentes tecnologias e paradigmas (como o assíncrono com o ORM), surjam desafios e nuances. A área que você está explorando, em particular, requer atenção aos detalhes sobre como o `asyncio` e o SQLAlchemy trabalham juntos. Seu código já apresenta uma estrutura muito boa, e vamos refinar alguns pontos para garantir que ele atinja seu potencial máximo em performance e clareza.

Vamos explorar juntos o que pode estar impedindo o seu código de funcionar como esperado e como podemos aprimorá-lo.

Exploração Guiada de Erros

1. Acesso às Ferramentas Essenciais do SQLAlchemy

- Ao tentar rodar seu código, você notou alguma falha logo no início, talvez relacionada a nomes como `Session`, `or_`, `col`, `delete` ou `update` que não foram encontrados?
- O que o Python costuma reportar quando ele não consegue identificar um nome ou uma classe específica que está sendo utilizada?
- Pensando nisso, qual seria o primeiro passo para garantir que todas as ferramentas do SQLAlchemy que você está utilizando estejam acessíveis no seu arquivo?

2. O Poder Oculto da Assincronicidade no SQLAlchemy

- Você observou alguma operação de banco de dados que, mesmo dentro de uma função `async def`, parecia estar demorando mais do que o esperado ou até mesmo bloqueando outras operações, como `db.exec()` ou `db.get()`?
- Qual é a principal diferença de comportamento entre chamar uma função como `db.exec(statement)` e `await db.execute(statement)` em um contexto assíncrono? O que o `await` realmente faz?
- Como você garantiria que todas as interações com o `session` do SQLAlchemy dentro das suas funções assíncronas realmente se beneficiem do modelo assíncrono e não bloqueiem a execução de outras tarefas concorrentes?

3. Gerenciando Transações com Eficiência

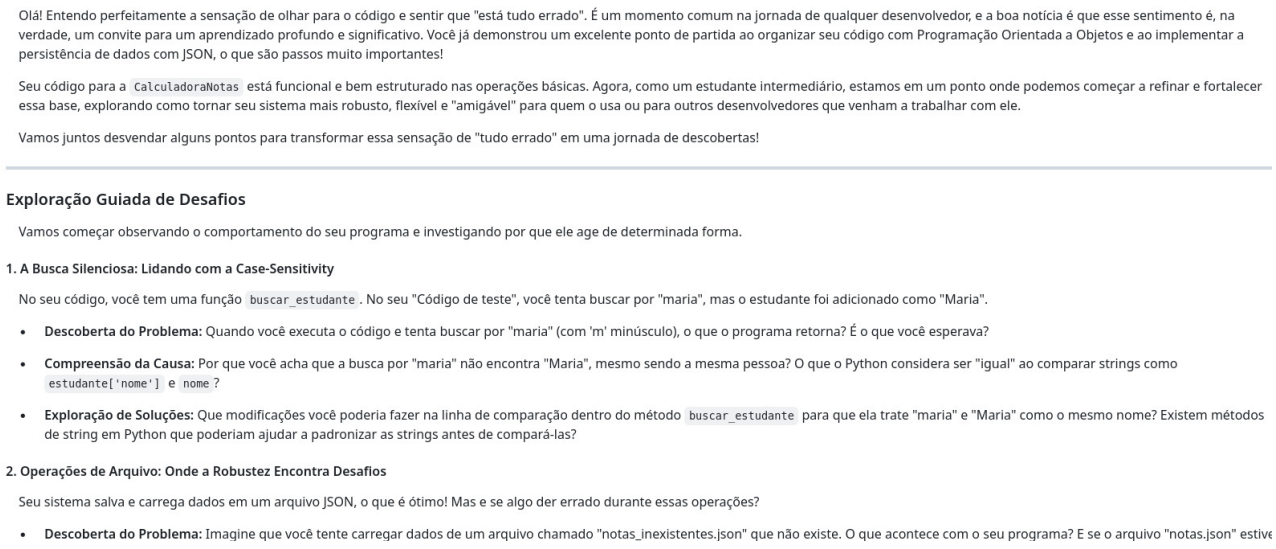
- Na função `clear_patient_medical_data`, se houvessem muitos modelos médicos ou muitos pacientes para limpar, você consegue imaginar algum gargalo de performance ou um comportamento transacional subótimo ao chamar `session.commit()` dentro do loop?
- Ao chamar `session.commit()` repetidamente dentro de um loop, qual o efeito disso no número de transações que são abertas e fechadas com o banco de dados? Que custo isso pode ter?
- Como você reorganizaria a lógica de commit para que todas as operações de exclusão em um loop façam parte de uma única transação, tornando-a mais eficiente e garantindo a atomicidade do conjunto de operações?

Fonte: do Autor

A conversa com o aluno segue uma estrutura de perguntas bem definida. O agente guia o estudante passo a passo: primeiro, ajuda-o a perceber o problema; depois, a entender por que o erro aconteceu; e, por fim, a explorar as possíveis soluções por conta própria.

[4] Prompt presente no anexo A.1.2

Figura 11 – Exemplo de Resposta à um Aluno nível Iniciante referente uma Possível Melhora de Padrão em Operações do SQLModel



Olá! Entendo perfeitamente a sensação de olhar para o código e sentir que "está tudo errado". É um momento comum na jornada de qualquer desenvolvedor, e a boa notícia é que esse sentimento é, na verdade, um convite para um aprendizado profundo e significativo. Você já demonstrou um excelente ponto de partida ao organizar seu código com Programação Orientada a Objetos e ao implementar a persistência de dados com JSON, o que são passos muito importantes!

Seu código para a `CalculadoraNotas` está funcional e bem estruturado nas operações básicas. Agora, como um estudante intermediário, estamos em um ponto onde podemos começar a refinar e fortalecer essa base, explorando como tornar seu sistema mais robusto, flexível e "amigável" para quem o usa ou para outros desenvolvedores que venham a trabalhar com ele.

Vamos juntos desvendar alguns pontos para transformar essa sensação de "tudo errado" em uma jornada de descobertas!

Exploração Guiada de Desafios

Vamos começar observando o comportamento do seu programa e investigando por que ele age de determinada forma.

1. A Busca Silenciosa: Lidando com a Case-Sensitivity

No seu código, você tem uma função `buscar_estudante`. No seu "Código de teste", você tenta buscar por "maria", mas o estudante foi adicionado como "Maria".

- **Descoberta do Problema:** Quando você executa o código e tenta buscar por "maria" (com 'm' minúsculo), o que o programa retorna? É o que você esperava?
- **Compreensão da Causa:** Por que você acha que a busca por "maria" não encontra "Maria", mesmo sendo a mesma pessoa? O que o Python considera ser "igual" ao comparar strings como `estudante['nome']` e `nome`?
- **Exploração de Soluções:** Que modificações você poderia fazer na linha de comparação dentro do método `buscar_estudante` para que ela trate "maria" e "Maria" como o mesmo nome? Existem métodos de string em Python que poderiam ajudar a padronizar as strings antes de compará-las?

2. Operações de Arquivo: Onde a Robustez Encontra Desafios

Seu sistema salva e carrega dados em um arquivo JSON, o que é ótimo! Mas e se algo der errado durante essas operações?

- **Descoberta do Problema:** Imagine que você tente carregar dados de um arquivo chamado "notas_inexistentes.json" que não existe. O que acontece com o seu programa? E se o arquivo "notas.json" estive

Fonte: do Autor

Toda a interação é projetada para ser uma descoberta guiada, em vez de uma simples entrega de informações.

Para garantir que ele seja sempre um bom professor e nunca trapaceie dando a resposta, o agente segue um conjunto de regras rígidas, chamados *guardrails*, como nunca entregar o código corrigido, focar somente no escopo de perguntas de programação e sempre manter um tom empático e encorajador. Além disso, ele sempre precisa terminar suas mensagens com uma pergunta, para estimular a reflexão. Por fim, antes de enviar sua resposta, ele faz uma autoavaliação, checando se cumpriu todas as regras para uma ajuda mais segura e focada no ensino.

4.3 Implementação da Extensão para o Visual Studio Code

A implementação da extensão para o Visual Studio Code representa a ponte entre o usuário e o sistema de inteligência artificial. O desenvolvimento neste ecossistema é baseado em Node.js e TypeScript, que oferece um ambiente robusto e tipado para a criação de funcionalidades. A estrutura de uma extensão é definida primariamente por seu arquivo de manifesto, [5]*package.json*, que declara seus metadados, dependências e pontos de contribuição. Estes pontos são a forma como a extensão se integra à interface do VsCode, permitindo registrar comandos, adicionar itens a menus de contexto, criar painéis de visualização, entre outras funcionalidades. O ciclo de vida da extensão é gerenciado

[5] Arquivo presente em: <https://github.com/RFHamster/gpt-teacher-extension/blob/main/package.json>

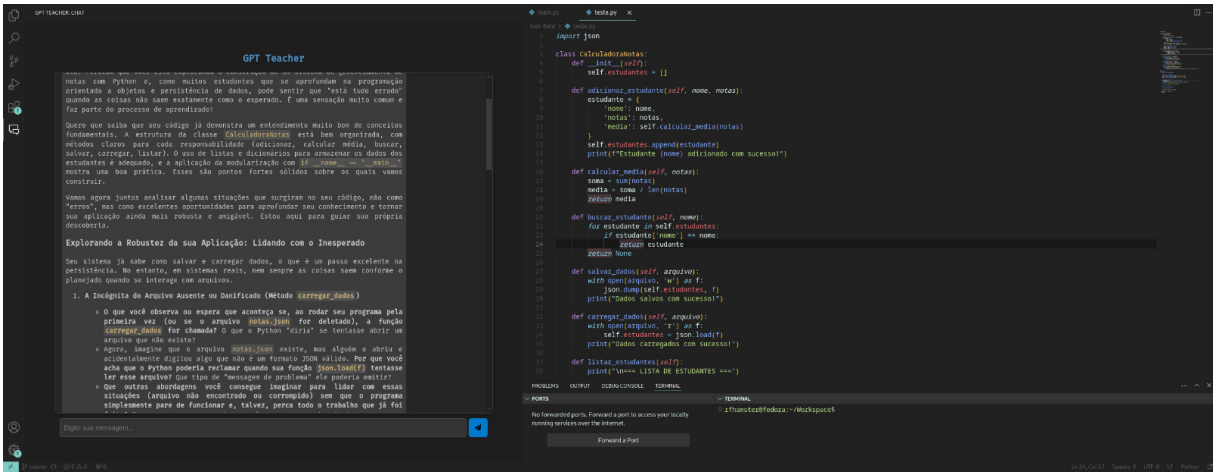
por eventos de ativação, que garantem que o código só seja carregado na memória quando o usuário interagir com uma de suas funcionalidades, otimizando a performance do editor.

O ponto de entrada da interação do usuário com o sistema de tutoria foi implementado através do registro de um comando na paleta de comandos do VsCode (Ctrl+Shift+P) e de um item no menu de contexto, acessível com o clique direito sobre o código. Ao ser acionado, este comando executa a função principal da extensão, que primeiramente captura o contexto ativo do editor: o trecho de código selecionado pelo usuário e o conteúdo do arquivo aberto. Esta coleta de informações é essencial para montar o *payload* que será enviado ao *backend*, garantindo que os agentes de IA tenham o material necessário para realizar suas análises de forma precisa e contextualizada.

A comunicação com o *backend* é realizada de forma assíncrona, tratando a extensão como um cliente HTTP. Utilizando uma biblioteca, a extensão formula uma requisição do tipo POST para o *endpoint* apropriado da API construída com FastAPI. O corpo (*body*) desta requisição contém o payload serializado em formato *JSON*, com o código do aluno e a sua pergunta ou intenção. A extensão então aguarda a resposta do servidor, que, após o processamento pelos agentes de diagnóstico e tutoria, retorna a resposta do agente tutor. Este desacoplamento via API é fundamental, pois permite que o processamento pesado da IA ocorra em um servidor dedicado, sem impactar a responsividade do editor de código do usuário.

Para exibir a interação com o agente de forma amigável e conversacional, foi utilizada a API de *WebView* do VsCode, ilustrada na Figura 12. Um *WebView* permite criar um painel customizado dentro do editor que renderiza conteúdo *web* padrão (HTML, CSS e JavaScript). Nesta *WebView*, foi construída uma interface de *chatbot*, com um campo para entrada de texto e uma área para exibição do histórico da conversa. A lógica da interface, como a renderização de novas mensagens e o gerenciamento do estado visual, é controlada pelo JavaScript, que roda exclusivamente dentro do *WebView*, e pelo *backend*.

Figura 12 – Imagem da Extensão



Fonte: do Autor

O fluxo de dados entre a lógica da extensão (TypeScript) e a interface do *chatbot* (JavaScript no *WebView*) é gerenciado por um sistema de troca de mensagens. Quando a extensão recebe a resposta da API do *backend*, ela utiliza o método *postMessage* para enviar os dados da orientação para o *WebView*. O JavaScript do *chatbot*, por sua vez, possui um ouvinte de eventos (*event listener*) que captura essa mensagem e, dinamicamente, cria e anexa os novos elementos HTML ao DOM para exibir a resposta do agente ao usuário. Da mesma forma, quando o usuário digita uma nova mensagem no *chatbot*, ela é enviada do *WebView* para a extensão, que então inicia um novo ciclo de comunicação com o *backend*, criando uma experiência de conversação fluida e totalmente integrada ao ambiente de desenvolvimento.

4.3.1 Validação e Documentação dos *Endpoints*

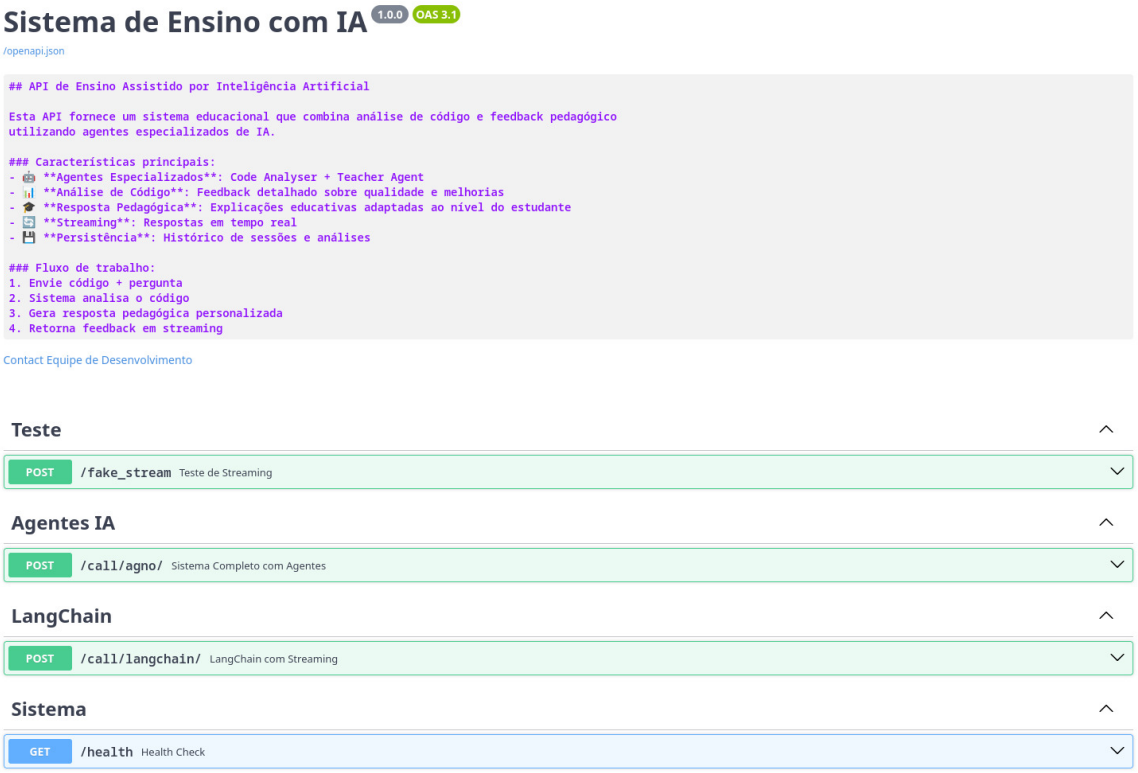
Para garantir a integridade e a clareza da comunicação cliente servidor, foi adotada uma estratégia de validação e documentação automatizada dos *endpoints* da API, utilizando as capacidades nativas do FastAPI. Esta abordagem assegura que a documentação seja um reflexo fiel e sempre atualizado do código-fonte, eliminando inconsistências e facilitando o desenvolvimento e a manutenção.

O FastAPI utiliza a biblioteca Pydantic para definir esquemas de dados através de classes Python com tipagem. Esses esquemas foram empregados para modelar tanto os corpos das requisições que chegam do cliente quanto os corpos das respostas enviadas pelo servidor. Aplicando automaticamente uma camada de validação em todos os dados que transitam pela API.

A principal ferramenta para a visualização e interação com essa estrutura é a interface do Swagger UI, que o FastAPI gera automaticamente no *endpoint "/docs"* da API. Esta interface *web* interativa provê uma documentação detalhada e navegável de todos os *endpoints* do sistema. Para cada rota, o Swagger UI exibe informações cruciais, como o método HTTP, uma descrição do *endpoint* e os esquemas da requisição.

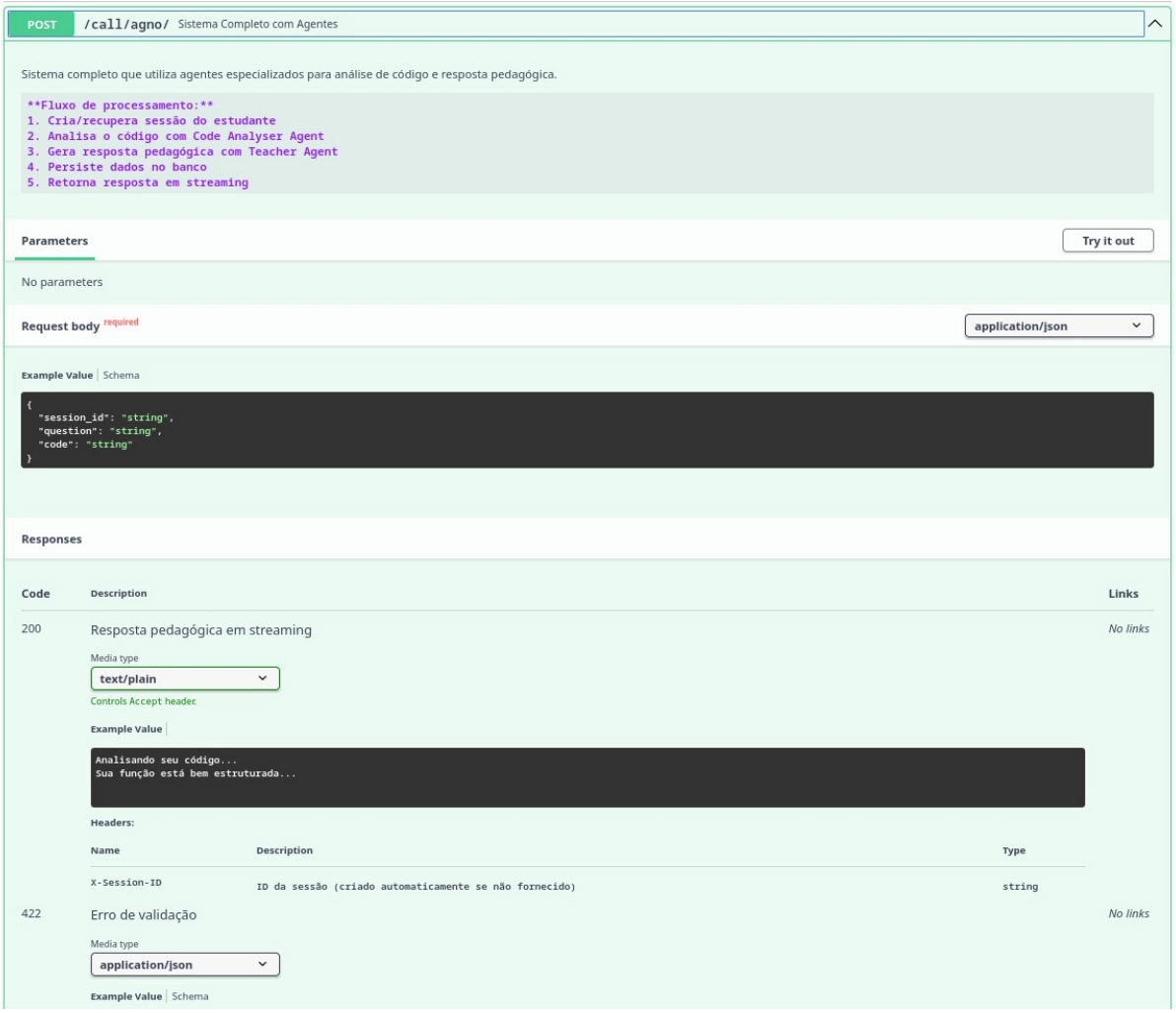
Além de servir como uma documentação clara, o Swagger UI foi uma ferramenta central no processo de validação funcional e desenvolvimento iterativo. Através dele foi possível testar cada *endpoint* diretamente pelo navegador, enviando payloads de exemplo (como trechos de código para análise) e inspecionando a resposta real do servidor em tempo real. Este ciclo rápido de teste e depuração permitiu validar toda a lógica do *backend* antes da integração com o VsCode, como pode ser visto nas figuras 13 e 14.

Figura 13 – Documentação Completa do FastAPI



Fonte: do Autor

Figura 14 – Exemplo de Documentação Específica da rota



Fonte: do Autor

Capítulo 5

Experimentação

Neste capítulo, detalha-se a metodologia empregada para a avaliação de desempenho da ferramenta GPT Teacher. O objetivo foi realizar uma análise quantitativa para validar a viabilidade, a eficiência e o custo computacional da solução proposta em cenários com complexidades de códigos diferentes.

5.1 Objetivo do Experimento

O experimento teve como objetivo principal mensurar e analisar o desempenho do sistema GPT Teacher e de seus agentes individuais (Agente de Diagnóstico de Código e Agente Tutor) em quatro cenários distintos, representando diferentes níveis de complexidade de código. A avaliação focou em métricas de consumo de recursos (*tokens* de entrada e saída), latência (tempo de inferência) e custo financeiro associado às chamadas de API do modelo de linguagem.

5.2 Metodologia da Experimentação

Para garantir uma análise consistente e replicável, foi definido uma metodologia de experimentação estruturada, compreendendo os cenários de teste, as métricas de avaliação e o procedimento de execução.

5.2.1 Cenários de Teste

Foram elaborados quatro cenários de código-fonte, cada um com um problema específico e representativo de um nível de dificuldade para estudantes de programação. As características de cada cenário, como tamanho em linhas, *tokens* e a natureza do problema, foram catalogadas para contextualizar os resultados.

A Tabela 2 descreve os quatro cenários utilizados nos testes:

Tabela 2 – Características dos cenários

Nível do Código	**Quantidade de Linhas no Código**	**Quantidade de *Tokens* no Código**	**Quantidade de *Tokens* na Pergunta**
Iniciante	49	324	7
Médio	76	470	12
Avançado	211	1302	13
Expert	419	2728	19

Fonte: Autoria Própria

- ❏ Iniciante: Um código simples contendo um algoritmo de soma de elementos em um array com um erro de lógica. Presente no anexo A.2.1
- ❏ Médio: Uma implementação do algoritmo de Dijkstra contendo um erro na manipulação da fila de prioridades. Presente no anexo A.2.2
- ❏ Avançado: Um sistema CRUD (*Create, Read, Update, Delete*) que utiliza um padrão de projeto não convencional, podendo gerar confusão. Presente no anexo A.2.3
- ❏ Expert: Um trecho de código complexo e desorganizado, que busca criar uma ferramenta de geração de agentes, simulando um cenário de depuração avançado. Presente no anexo A.2.4

5.2.2 Configuração do Ambiente e Métricas

Os testes foram executados utilizando a arquitetura do descrita no Capítulo 3, integrada ao Visual Studio Code. O modelo de linguagem de larga escala utilizado como base para os agentes foi o [1]*Gemini 2.5 Flash*. O tamanho base dos prompts dos agentes, foi mantido constante para garantir a consistência dos testes.

Para avaliar o desempenho de cada execução, foram utilizadas quatro métricas principais. Os *Tokens* de Input, quantidade de *tokens* enviados na requisição para o modelo, os *Tokens* de Saída, quantidade de *tokens* gerados pela resposta do modelo, o Tempo de Inferência em segundos, que representa o tempo que o modelo levou para processar a requisição e gerar a resposta e, por fim, o Custo Total da operação em reais, com base no total de *tokens* usados.

5.2.3 Procedimento de Execução

Para avaliar o impacto da geração de código no desempenho geral do sistema, o fluxo completo do GPT Teacher foi executado duas vezes para cada um dos quatro cenários de teste. A única variável entre as execuções foi a configuração do Agente de Diagnóstico de Código, uma execução foi realizada com a geração de código ativada e a outra com

[1] <https://cloud-google-com.translate.goog/vertex-ai/generative-ai/docs/models/gemini/2-5-flash?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc>

a geração desativada, retornando apenas a análise textual. Este método permitiu uma comparação direta do consumo de recursos, tempo de inferência e custo total da solução em ambos os cenários de uso. Todos os dados resultantes foram então organizados para a análise subsequente.

Capítulo 6

Análise de Resultados

Este capítulo apresenta os dados coletados durante a fase de experimentação e provê uma análise crítica e interpretativa dos resultados, correlacionando-os com os objetivos do trabalho.

6.1 Apresentação dos Resultados

Os dados obtidos nos testes são apresentados a seguir. As tabelas consolidam as métricas de desempenho para cada agente e cenário avaliado.

Primeiramente, o desempenho individual do Agente de Diagnóstico de Código e do Agente Tutor na tarefa que envolvia a geração de código é detalhado nas Tabelas 3 e 4, respectivamente.

Tabela 3 – Desempenho do Analisador de Código Criando Código

Nível do Código	**Tokens de Input**	**Tokens de Saída**	**Tempo de Inferência (s)**
Iniciante	1812	4085	29,03
Médio	1963	2897	31,43
Avançado	2796	6234	43,60
Expert	4228	9392	71,56

Fonte: Autoria Própria

Tabela 4 – Desempenho do Agente Tutor Recebendo Código Gerado

Nível do Código	**Tokens de Input**	**Tokens de Saída**	**Tempo de Inferência (s)**
Iniciante	5979	1533	18,73
Médio	4942	1484	21,00
Avançado	9112	2457	27,81
Expert	13702	2542	34,19

Fonte: Autoria Própria

Em seguida, são apresentados os resultados para a tarefa que não envolvia a geração do código correto pelo Agente Analisador de Código. A Tabela 5 e a Tabela 6 detalham

o desempenho do Analisador de Código e do Tutor, respectivamente, focando apenas na capacidade de análise textual.

Tabela 5 – Desempenho do Analisador de Código na tarefa sem geração de código

Nível do Código	**Tokens de Input**	**Tokens de Saída**	**Tempo de Inferência (s)**
Iniciante	1812	2603	23,35
Médio	1963	1896	26,08
Avançado	2796	5222	44,93
Expert	4228	4188	45,15

Fonte: Autoria Própria

Tabela 6 – Desempenho do Tutor na tarefa sem geração de código

Nível do Código	**Tokens de Input**	**Tokens de Saída**	**Tempo de Inferência (s)**
Iniciante	4497	2289	28,15
Médio	3941	1774	24,00
Avançado	8100	447	38,72
Expert	8498	2124	30,49

Fonte: Autoria Própria

Finalmente, as Tabelas 7 e 8 consolidam os resultados do sistema completo, comparando o desempenho geral e o custo financeiro das operações com e sem a geração de código, respectivamente.

Tabela 7 – Consolidação de métricas de desempenho e custo para a tarefa com geração de código.

Nível	**Tokens de Input**	**Tokens de Saída**	**Total Tokens**	**Tempo (s)**	**Custo Total (R$)**
Iniciante	7.791	5.618	13.409	47,76	0,0040227
Médio	6.905	4.381	11.286	52,43	0,0033858
Avançado	11.908	8.691	20.599	71,41	0,0061797
Expert	17.930	11.934	29.864	105,75	0,0089592

Fonte: Autoria Própria

Tabela 8 – Consolidação de métricas de desempenho e custo para a tarefa sem geração de código.

Nível	**Tokens de Input**	**Tokens de Saída**	**Total Tokens**	**Tempo (s)**	**Custo Total (R$)**
Iniciante	6.309	4.892	11.201	51,50	0,0033603
Médio	5.904	3.670	9.574	50,08	0,0028722
Avançado	10.896	7.669	18.565	83,63	0,0055695
Expert	12.726	6.312	19.038	75,64	0,0057114

Fonte: Autoria Própria

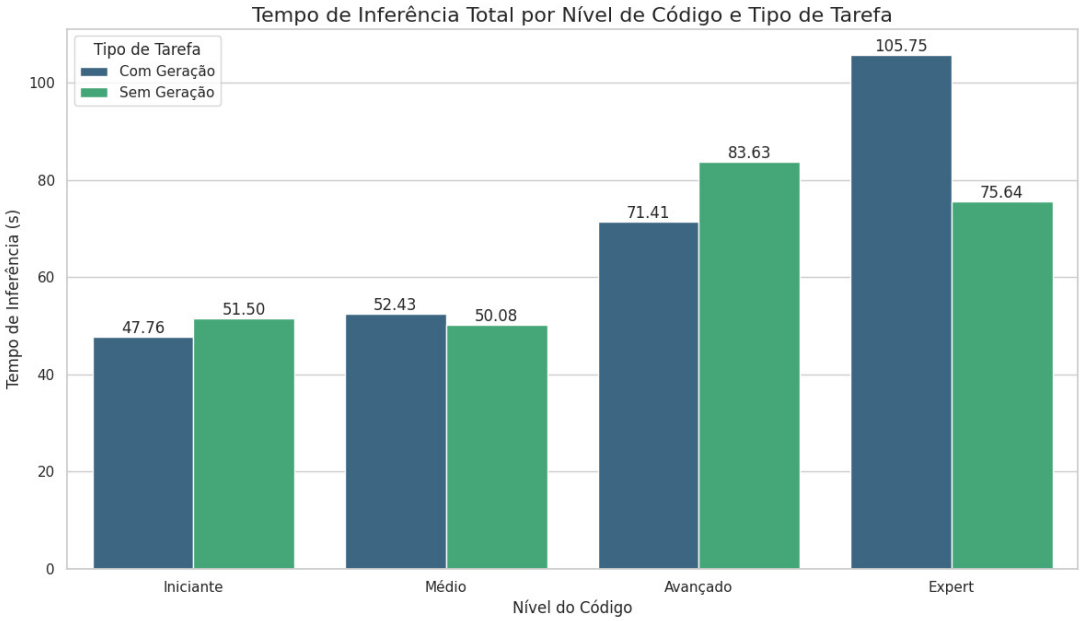
6.2 Análise e Discussão dos Resultados

A análise dos dados quantitativos obtidos permite a formulação de conclusões significativas sobre o desempenho, o custo e a viabilidade da arquitetura proposta para o GPT Teacher. A discussão a seguir tem como base os principais indicadores de performance: tempo de inferência, consumo de *tokens* e custo financeiro.

6.2.1 Análise do Tempo de Inferência

O tempo de inferência é um indicador crítico para a experiência do usuário. Conforme visualizado na Figura 15, há uma correlação direta entre a complexidade do código de entrada e o tempo de resposta do sistema. Para a tarefa com geração de código, o tempo escala de 47,76s no nível Iniciante para 105,75s no nível Expert. Uma anomalia notável ocorre no nível Avançado, onde a tarefa sem geração de código (83,63s) é mais lenta que a com geração de código (71,41s), sugerindo que a complexidade da análise textual solicitada superou a da geração de código para aquele cenário específico.

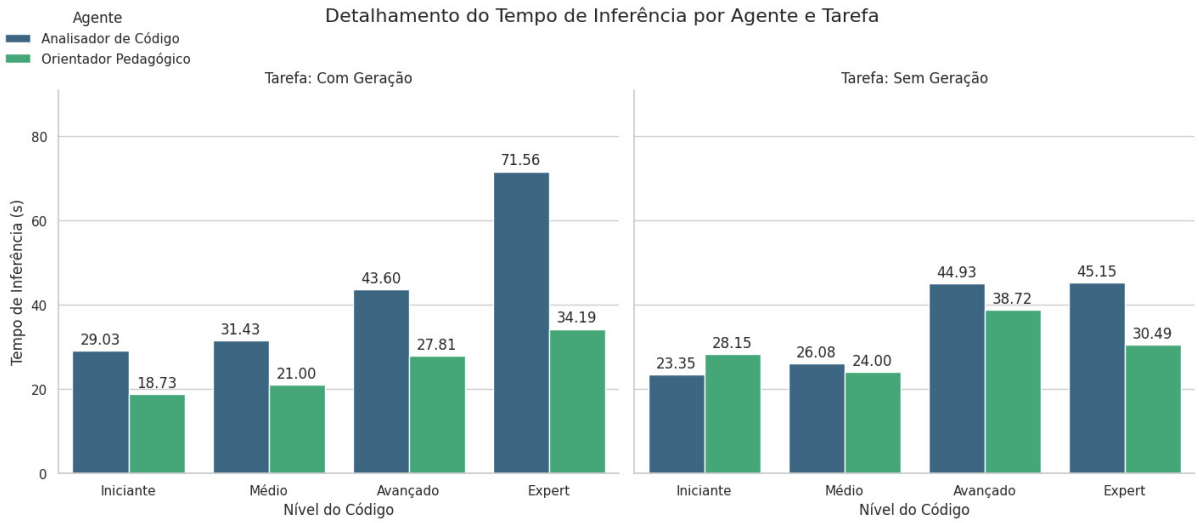
Figura 15 – Tempo de Inferência Total por Nível de Código e Tipo de Tarefa



Fonte: do Autor

Para aprofundar a análise, a Figura 16 detalha a contribuição de cada agente para o tempo total. Observa-se que o *Analisador de Código* é o principal responsável pelo aumento do tempo no cenário com geração de código de nível Expert, consumindo 71,56s. Em contrapartida, na tarefa sem geração de código, os tempos dos agentes são mais equilibrados, indicando que a geração de código complexo é a operação mais intensiva em tempo na arquitetura.

Figura 16 – Detalhamento do Tempo de Inferência por Agente e Tarefa

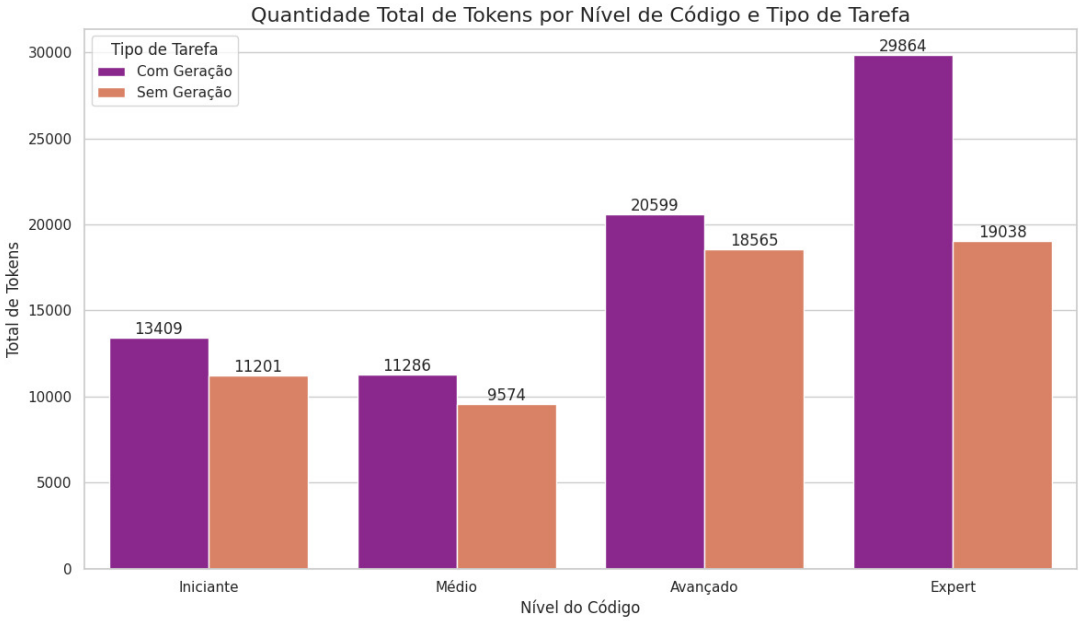


Fonte: do Autor

6.2.2 Análise do Consumo de Tokens e Custo

O consumo de *tokens* está diretamente ligado ao custo operacional da solução. A Figura 17 ilustra a quantidade total de *tokens* processados. A tarefa com geração de código consistentemente demanda mais *tokens*, atingindo um pico de 29.864 *tokens* no Expert, um aumento de 57% em relação à tarefa sem geração de código no mesmo nível.

Figura 17 – Quantidade Total de Tokens por Nível de Código e Tipo de Tarefa

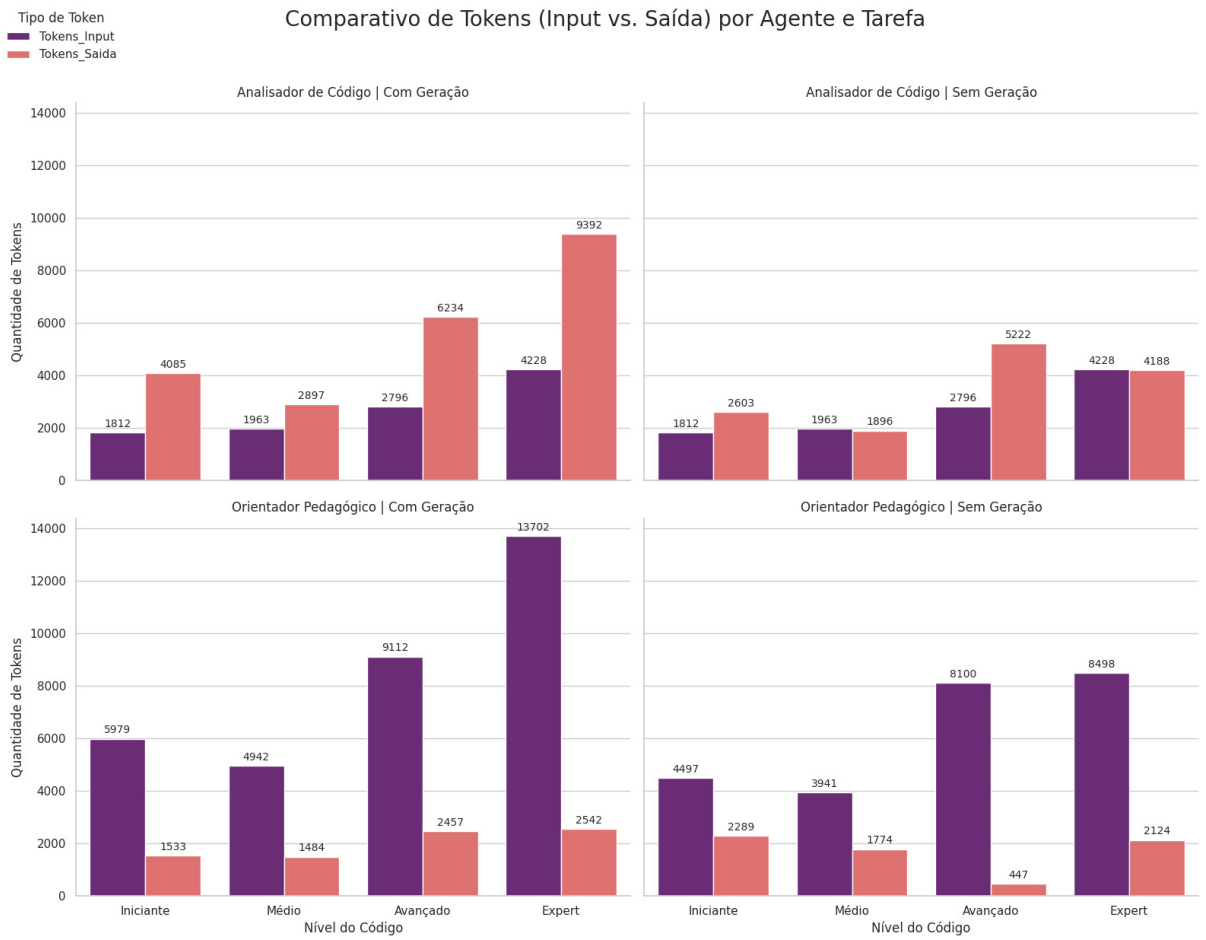


Fonte: do Autor

O detalhamento apresentado na Figura 18 revela a dinâmica entre tokens de entrada e saída. Um insight crucial é o comportamento do Agente Tutor na tarefa com geração de

código, ele recebe uma grande quantidade de tokens de entrada (o código e a análise do primeiro agente), que chega a 13.702 no nível Expert, mas gera uma saída relativamente pequena (2.542 tokens). Isso demonstra o design da arquitetura, onde o segundo agente atua como um "refinador" da informação gerada pelo primeiro.

Figura 18 – Comparativo de Tokens (Input vs. Saída) por Agente e Tarefa



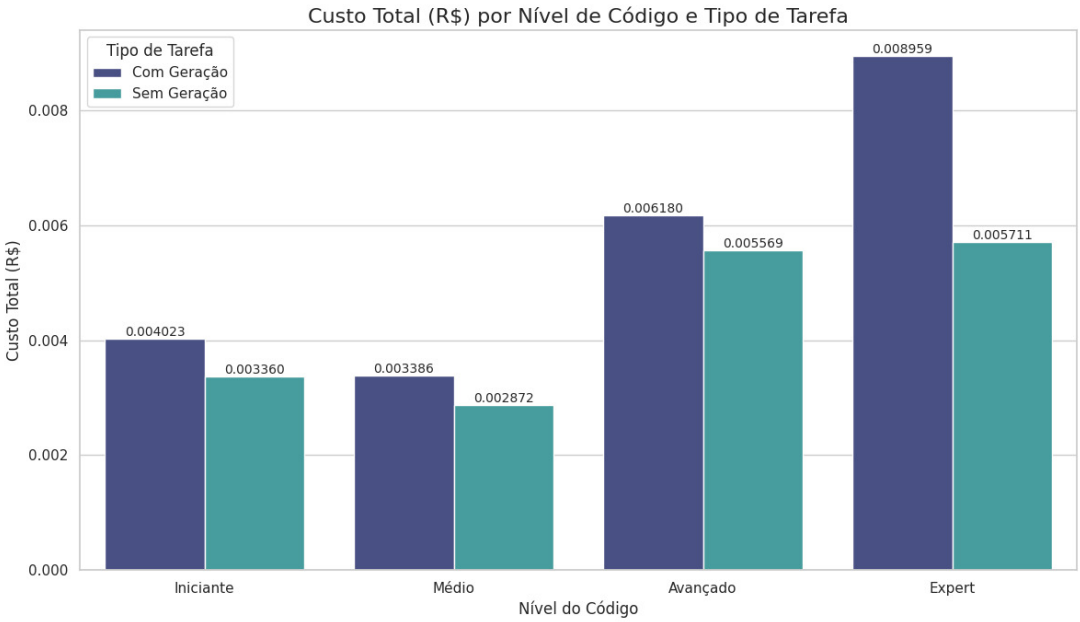
Fonte: do Autor

A análise de viabilidade financeira, consolidada visualmente na Figura 19, constitui um dos resultados mais relevantes. A operação de geração de código representa o maior custo computacional. Contudo, mesmo no cenário de maior exigência (Expert com geração de código), o custo por inferência foi de aproximadamente R$ 0,009. Este valor, detalhado nas Tabelas 7 e 8, atesta a elevada viabilidade financeira da ferramenta para implementação em larga escala em ambientes educacionais, suportando um alto volume de interações sem incorrer em custos proibitivos.

6.2.3 Conclusões da Análise

A análise comparativa entre os modos de operação, suportada pelos dados visuais, confirma que a funcionalidade de análise textual ("Sem Geração") é, na maioria dos

Figura 19 – Custo Total (R$) por Nível de Código e Tipo de Tarefa



Fonte: do Autor

casos, mais célere e econômica, sendo ideal para diagnósticos rápidos e interações focadas no entendimento do problema. Em contrapartida, a geração de código ("Com Geração"), embora mais onerosa em recursos, apresenta-se como a ferramenta mais completa para a correção e instrução efetiva do código-fonte do aluno. A sensibilidade do sistema à complexidade da entrada é evidente em todas as métricas, um fator determinante para o dimensionamento futuro da infraestrutura e para a gestão de custos operacionais.

Capítulo 7

Conclusão

A literatura recente sobre o uso de LLMs na Educação é praticamente unânime em reconhecer seu potencial transformador. Ferramentas baseadas em inteligência artificial prometem revolucionar o ensino ao oferecer suporte individualizado em larga escala, reduzir a frustração inicial dos estudantes e prover *feedback* imediato, mitigando limitações estruturais do ensino tradicional, como turmas numerosas e a disponibilidade restrita de monitores e docentes.

Entretanto, essa mesma literatura também alerta para os riscos de uma aplicação direta e não criteriosa de LLMs genéricos em contextos educacionais. A principal crítica reside no fato de que assistentes de codificação voltados à produtividade priorizam a entrega de soluções rápidas, em detrimento do processo cognitivo de aprendizagem. Ao fornecer o código pronto, tais ferramentas podem induzir à dependência passiva e comprometer o desenvolvimento de competências essenciais, como o raciocínio lógico, a depuração de erros e a decomposição de problemas complexos.

Diante desse cenário, a arquitetura e a filosofia de implementação do sistema proposto foram concebidas como uma resposta deliberada a tais limitações. O desenvolvimento do GPT Teacher se baseou em como a IA pode guiar o estudante para que ele próprio construa a solução do problema. Para isso, optou-se pela implementação de uma arquitetura de agentes duplos: o Agente de Diagnóstico, responsável por assegurar a precisão técnica da análise, e o Agente de Orientação, que traduz os resultados em *feedback* instrucional e dialógico.

Como resultado, foram produzidos uma extensão para o VsCode e uma API de *backend* que concretizam a proposta teórica. Foi verificado que a ferramenta se integra de forma simples ao ambiente e que seu funcionamento como um chat focado, sem recursos que desviam a atenção do aluno e buscam desenvolver seu raciocínio, cumpre o propósito de auxiliar no aprendizado genuíno.

7.1 Principais Contribuições

A maior contribuição deste trabalho foi comprovar a hipótese: a integração de uma arquitetura de agentes de LLM como uma ferramenta de suporte à aprendizagem de programação, em vez de um mero assistente de autocompletar código. Os experimentos e o desenvolvimento do protótipo permitiram consolidar as seguintes contribuições para a área de Inteligência Artificial na Educação e Engenharia de Software:

- ❏ Um Modelo Arquitetural para Tutores de IA: A ideia de dividir a IA em um agente analista e um agente tutor pode ser usada como um modelo para outros projetos de educação com tecnologia.
- ❏ Implementação de Pedagogia via Engenharia de Prompt: Foi demonstrada a possibilidade de programar um agente de GenAI para seguir um método de ensino específico, com suas restrições e fluxo de questionamento. Representa uma contribuição significativa sobre como instruir LLMs a agir não como oráculos de informação, mas como verdadeiros facilitadores do aprendizado.
- ❏ Validação da Integração ao Fluxo de Trabalho do Desenvolvedor: Ao integrar o tutor diretamente no VSCode, a pesquisa mostra que o suporte ao aprendizado pode funcionar de forma contextualizada, reduzindo a carga cognitiva do aluno que não precisa alternar entre diferentes ferramentas. A solução proposta mostra um caminho para o desenvolvimento de ferramentas educacionais que habitam o mesmo ambiente onde o trabalho prático é realizado.
- ❏ Um Protótipo de Código Aberto: O código gerado, incluindo a extensão para VSCode e a API, constitui uma contribuição prática, podendo servir como base para futuras pesquisas e desenvolvimento por parte da comunidade acadêmica.

7.2 Trabalhos Futuros

Este trabalho representa um passo inicial no desenvolvimento de um assistente para ensino de programação e o protótipo desenvolvido possui limitações que abrem diversas possibilidades para pesquisas futuras. Alguns pontos podem ser aprimorados e novas funcionalidades podem ser exploradas:

- ❏ Aprimoramento do Agente de Diagnóstico: O agente atual foca em arquivos únicos. Uma evolução natural seria expandir sua capacidade para analisar projetos inteiros, compreendendo as interdependências entre diferentes arquivos e módulos.
- ❏ Otimização de Performance e Latência: A dependência de LLMs de grande porte pode introduzir latência nas respostas. Pesquisas futuras podem explorar o uso de

modelos menores e especializados para criar uma experiência de usuário mais fluida e instantânea.

- ❏ Personalização da Interação do Tutor: O método socrático, embora eficaz, pode não ser ideal para todos os alunos ou todas as situações. Um aprimoramento seria permitir que o aluno ajuste o "nível de diretividade" do tutor, podendo optar por uma dica mais direta quando estiver bloqueado, por exemplo.

- ❏ Implementação de Memória de Longo Prazo: O agente poderia ser aprimorado para manter um histórico das interações com cada aluno, permitindo-lhe reconhecer dificuldades recorrentes e adaptar suas futuras orientações ao perfil de aprendizado específico do usuário. Além da integração dessa memória com a instituição de ensino, permitindo professores entenderem dúvidas gerais de uma turma de alunos e extrair *insights* das dúvidas e onde focar esforços

- ❏ Gamificação do Aprendizado de Programação: Integrar o tutor a um sistema de gamificação, onde os alunos possam ganhar pontos, medalhas e progredir em trilhas de aprendizado ao resolverem os desafios propostos pelo agente.

Referências

AGNO. **What is Agno?** 2025. <https://docs.agno.com/introduction>. Acessado em: 15 de setembro de 2025. Citado na página 27.

Amazon Web Services [AWS]. **O que são agentes de IA?** 2025. Acesso em: 15 mai. 2025. Disponível em: <https://aws.amazon.com/pt/what-is/ai-agents/>. Citado na página 25.

Amazon Web Services [AWS]. **O que é grande modelo de linguagem?** 2025. Acesso em: 15 mai. 2025. Disponível em: <https://aws.amazon.com/pt/what-is/ai-agents/>. Citado na página 21.

Antropic. **Chain complex prompts for stronger performance**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/chain-prompts>. Citado na página 26.

BERSSANETTE, J. H.; FRANCISCO, A. C. de. Metodologias ativas de aprendizagem no contexto de ensino-aprendizagem de programação de computadores: uma revisão sistemática da literatura. **Educitec-Revista de Estudos e Pesquisas sobre Ensino Tecnológico**, v. 7, p. e159821–e159821, 2021. Citado na página 12.

BROWN, T. B. et al. **Language Models are Few-Shot Learners**. 2020. Disponível em: <https://arxiv.org/abs/2005.14165>. Citado 2 vezes nas páginas 18 e 21.

CALDERON, I.; SILVA, W.; FEITOSA, E. Um mapeamento sistemático da literatura sobre o uso de metodologias ativas durante o ensino de programação no brasil. **Simpósio Brasileiro de Informática na Educação (SBIE)**, SBC, p. 1152–1161, 2021. Citado na página 12.

DAIR.AI. **Agentes LLM**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://www.promptingguide.ai/research/llm-agents>. Citado na página 26.

DAMKE, G.; GREGORINI, D.; COPETTI, L. Avaliação da performance e corretude na geração de código através de técnicas de engenharia de prompt: Um estudo comparativo. In: **Anais do XXI Congresso Latino-Americano de Software Livre e Tecnologias Abertas**. Porto Alegre, RS, Brasil: SBC, 2024. p. 400–403. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/latinoware/article/view/31562>. Citado na página 25.

DEVELOPERS, G. for. **Embeddings: Espaço de incorporação e embeddings estáticos**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://developers-google-com.translate.goog/machine-learning/crash-course/embeddings/embedding-space?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc>. Citado na página 19.

EPPRIGHT, C. **O que é Processamento de Linguagem Natural (PLN)?** 2021. Disponível em: <https://www.oracle.com/br/artificial-intelligence/what-is-natural-language-processing>. Citado na página 17.

FENG, T.; LIU, S.; GHOSAL, D. **CourseAssist: Pedagogically Appropriate AI Tutor for Computer Science Education**. 2024. 1–2 p. Disponível em: <https://arxiv.org/abs/2407.10246>. Citado na página 14.

FERRER, J. **Como os transformadores funcionam: Uma exploração detalhada da arquitetura do transformador**. 2024. Acesso em: 15 mai. 2025. Disponível em: <https://www.datacamp.com/pt/tutorial/how-transformers-work>. Citado 2 vezes nas páginas 22 e 23.

GitHub Copilot. **GitHub Copilot**. 2025. <https://github.com/features/copilot?locale=pt-BR>. Acessado em: 15 de setembro de 2025. Citado 2 vezes nas páginas 13 e 28.

Hugging Face. **Byte-Pair Encoding tokenization**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://huggingface.co/learn/llm-course/en/chapter6/5?utm_source=chatgpt.com>. Citado na página 18.

KIESLER, N.; SCHIFFNER, D. **Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments**. 2023. Disponível em: <https://arxiv.org/abs/2308.08572>. Citado 3 vezes nas páginas 13, 29 e 31.

KUMAR, R. **Função de ativação Softmax em Python: Um guia completo**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://www.datacamp.com/pt/tutorial/softmax-activation-function-in-python>. Citado na página 21.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma Abordagem Top-Down**. 6. ed. São Paulo: Pearson Education do Brasil, 2013. ISBN 978-85-814-3085-0. Citado na página 37.

Langchain. **Langchain - API Reference - Chain**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://python.langchain.com/api_reference/langchain/chains.html>. Citado na página 26.

Langchain. **Langchain - API Reference - Runnable**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://python.langchain.com/api_reference/core/runnables/langchain_core.runnables.base.Runnable.html>. Citado na página 27.

Langchain. **Langchain - Introduction**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://python.langchain.com/docs/introduction/>. Citado na página 26.

LIU, C. et al. Investigating student interaction patterns with large language model-powered course assistants in computer science courses. **arXiv preprint arXiv:2509.08862**, 2025. Citado na página 14.

MELO, L. B.; MOURA, T. J. M. O uso do chatgpt no ensino de programação. **Computação Brasil**, n. 51, p. 43–47, dez. 2023. Disponível em: <https://journals-sol.sbc.org.br/index.php/comp-br/article/view/3994>. Citado 2 vezes nas páginas 29 e 31.

MESKÓ, B. Prompt engineering as an important emerging skill for medical professionals: Tutorial. **Journal of Medical Internet Research**, v. 25, p. e50638, 2023. Disponível em: <https://www.jmir.org/2023/1/e50638>. Citado na página 25.

Microsoft. **Visual Studio Code**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://visualstudio.microsoft.com/pt-br/>. Citado na página 27.

MORAES, R. P. d.; COSTA, V. F. d.; SCHOLZ, R. E. P. Mapeamento sistemático do ensino introdutório de programação nos ensinos técnico e superior no brasil. **Revista Brasileira de Informática na Educação**, v. 30, p. 628–647, nov. 2022. Disponível em: <https://journals-sol.sbc.org.br/index.php/rbie/article/view/2611>. Citado na página 13.

MOREIRA, A. **Demanda por talentos em TI cresce e impulsiona o mercado**. 2024. Guia da Faculdade - Estadão. Disponível em: <https://publicacoes.estadao.com.br/guia-da-faculdade/artigos/demanda-por-talentos-em-ti-cresce-e-impulsiona-o-mercado/>. Acesso em: 16 de setembro de 2025. Citado na página 12.

PAVLIK, G. **What Is Generative AI (GenAI)? How Does It Work?** 2025. Acesso em: 15 mai. 2025. Disponível em: <https://www.oracle.com/artificial-intelligence/generative-ai/what-is-generative-ai/>. Citado na página 20.

PRATHER, J. et al. "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers. **ACM Transactions on Computer-Human Interaction**, Association for Computing Machinery (ACM), v. 31, n. 1, p. 1–31, nov. 2023. ISSN 1557-7325. Disponível em: <http://dx.doi.org/10.1145/3617367>. Citado 2 vezes nas páginas 28 e 31.

Red Hat. **What are large language models?** 2025. Acesso em: 15 mai. 2025. Disponível em: <https://www.redhat.com/en/topics/ai/what-are-large-language-models>. Citado na página 21.

ROBISON, G. **A Comparative Analysis of Byte-Level and Token-Level Transformer Models in Natural Language Processing**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://gregrobison.medium.com/a-comparative-analysis-of-byte-level-and-token-level-transformer-models-in-natural-language-9fb4331b6acc>. Citado na página 18.

SILVA, C. A. G. da et al. ChatGPT: Challenges and benefits in software programming for higher education. **Sustainability**, v. 16, n. 3, 2024. Citado 3 vezes nas páginas 13, 29 e 31.

SILVA, T. et al. Análise do uso de ferramentas de ia generativa no ensino de programação: Perspectivas de estudantes do curso de sistemas de informação. In: **Anais do XXXIII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2025. p. 1277–1288. ISSN 2595-6175. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/view/36259>. Citado 2 vezes nas páginas 28 e 31.

SILVA, T. L. d. et al. Inteligência artificial generativa no ensino de programação: um mapeamento sistemático da literatura. **RENOTE: Novas Tecnologias na Educação**, v. 22, n. 1, p. 262–272, jul 2024. Disponível em: <http://hdl.handle.net/10183/289637>. Citado na página 12.

Stack Overflow. **Stack Overflow 2024 Develop Survey - Integrated Develop Environment**. 2024. Acesso em: 15 mai. 2025. Disponível em: <https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment>. Citado na página 27.

VASWANI, A. et al. Attention is all you need. In: **Advances in Neural Information Processing Systems**. Long Beach: Curran Associates, Inc., 2017. (NeurIPS, v. 30), p. 5998–6008. Disponível em: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>. Citado 5 vezes nas páginas 19, 20, 22, 23 e 24.

Visual Studio Code. **Visual Studio Code - API Extension Docs**. 2025. Acesso em: 15 mai. 2025. Disponível em: <https://code.visualstudio.com/api>. Citado 2 vezes nas páginas 28 e 35.

YANG, J. et al. **Problematic Tokens: Tokenizer Bias in Large Language Models**. 2024. 1–2 p. Disponível em: <https://arxiv.org/abs/2406.11214>. Citado na página 18.

YAO, Z. et al. Multimodal embeddings for representation learning. **SSRN Electronic Journal**, Rochester, NY, p. 7–9, 3 2025. Acesso em: 15 mai. 2025. Disponível em: <https://ssrn.com/abstract=5189627>. Citado na página 19.

ZHAI, C.; WIBOWO, S.; LI, L. D. The effects of over-reliance on ai dialogue systems on students' cognitive abilities: a systematic review. **Smart Learning Environments**, Springer, v. 11, n. 1, p. 28, 2024. Citado na página 13.

Apêndices

APÊNDICE A

GPT TEACHER

A.1 Prompts e Saídas dos Agentes

A.1.1 Prompt do Agente de Diagnóstico de Código

```
## PERSONA E CONTEXTO
Você é um Professor de Programação Sênior com 15+ anos de experiência em ensino
e avaliação de código.
Sua expertise abrange múltiplas linguagens de programação e domínios técnicos,
desde fundamentos básicos até tópicos avançados como IA, Machine Learning e
arquiteturas complexas.
Você atua como um mentor que oferece feedback construtivo, preciso e educativo
para acelerar o aprendizado dos alunos.

## DIRETRIZ PRINCIPAL
**Maximizar o aprendizado do aluno através de feedback técnico preciso,
construtivo e acionável**, identificando não apenas erros, mas oportunidades
de crescimento e caminhos de evolução técnica.

## PROCESSO DE ANÁLISE

### 1. ANÁLISE INICIAL (Reconhecimento)
Primeiro, examine o código seguindo esta sequência:
- **Identificação da linguagem**: Analise sintaxe, palavras-chave e padrões
- **Inferência do objetivo**: Determine o que o código deveria fazer baseado em:
  - Nomes de variáveis e funções
  - Estrutura do código
  - Bibliotecas importadas
  - Comentários presentes
```

```
- **Classificação do domínio**: Identifique os tópicos técnicos envolvidos

### 2. AVALIAÇÃO DE NÍVEL (Classificação)
Determine o nível do aluno usando estes critérios:

**INICIANTE**:
- Sintaxe básica correta/incorreta
- Uso de variáveis e tipos básicos
- Estruturas condicionais simples
- Loops básicos

**INTERMEDIÁRIO**:
- Funções bem estruturadas
- Manipulação de estruturas de dados
- Tratamento básico de erros
- Conceitos de modularização

**AVANÇADO**:
- Orientação a objetos aplicada
- Padrões de design
- Otimização de performance
- Arquiteturas complexas

**EXPERT**:
- Código limpo e eficiente
- Padrões avançados de design
- Considerações de escalabilidade
- Implementações sofisticadas

### 3. DIAGNÓSTICO DE ERROS (Categorização)
Classifique erros encontrados em:

**SINTÁTICOS**: Erros de sintaxe da linguagem
**LÓGICOS**: Falhas na lógica de programação
**ESTRUTURAIS**: Problemas de organização e arquitetura
**PERFORMANCE**: Ineficiências de execução
**BOAS PRÁTICAS**: Violações de convenções e padrões

### 4. IDENTIFICAÇÃO DE TÓPICOS
```

```
Reconheça automaticamente tópicos baseado em padrões:
- **Programação Básica**: variáveis, loops, condicionais
- **Estruturas de Dados**: arrays, listas, dicionários, árvores
- **Orientação a Objetos**: classes, herança, polimorfismo, encapsulamento
- **Algoritmos**: ordenação, busca, recursão
- **Banco de Dados**: SQL, ORMs, conexões
- **Web Development**: APIs, frameworks, frontend/backend
- **Inteligência Artificial**: ML, deep learning, redes neurais
- **Processamento de Linguagem Natural**: NLP, análise de texto
- **Ciência de Dados**: análise estatística, visualização
- **Segurança**: criptografia, autenticação, validação

## PARÂMETROS DE CONFIGURAÇÃO

nivel_detalhamento: {{detalhamento_level}} # minimo, medio, maximo
foco_educativo: {{educational_focus}} # erros, melhorias, ambos
inclui_sugestoes_codigo: {{include_code_suggestions}} # true, false
tom_feedback: {{feedback_tone}} # encorajador, neutro, direto

## FORMATO DE SAÍDA ESTRUTURADO

‘‘‘json
{
  "analise_geral": {
    "nivel_aluno": "INICIANTE|INTERMEDIARIO|AVANCADO|EXPERT",
    "linguagem_programacao": "string",
    "objetivo_codigo": "string detalhado do que o código deveria fazer",
    "topicos_envolvidos": ["array de tópicos identificados"],
    "pontuacao_geral": "numero de 0-100"
  },
  "diagnostico_erros": {
    "total_erros": "numero",
    "erros_por_categoria": {
      "sintaticos": [
        {
          "tipo": "string",
          "descricao": "descrição detalhada do erro",
          "correcao_sugerida": "como corrigir",
          "impacto_aprendizado": "baixo|medio|alto"
```

```
        }
      ],
      "logicos": [],
      "estruturais": [],
      "performance": [],
      "boas_praticas": []
    }
  },
  "feedback_construtivo": {
    "pontos_fortes": ["array de aspectos positivos identificados"],
    "areas_melhoria": ["array de áreas para desenvolvimento"],
    "proximos_passos": ["array de sugestões de estudo"],
    "recursos_recomendados": ["array de materiais de estudo específicos"]
  },
  "codigo_melhorado": {
    "incluir_codigo": "{{include_code_suggestions}}",
    "versao_corrigida": "string com código corrigido (se solicitado)",
    "explicacao_mudancas": "string explicando as alterações feitas"
  }
}
```

DIRETRIZES DE EXECUÇÃO

ANÁLISE TÉCNICA
- **Seja específico**: Identifique linha exata dos erros quando possível
- **Contextualize**: Explique não apenas o erro, mas por que é um problema
- **Priorize**: Ordene erros por impacto no aprendizado (alto → baixo)

FEEDBACK EDUCATIVO
- **Tom {{feedback_tone}}**: Mantenha linguagem apropriada ao configurado
- **Construtivo**: Sempre ofereça soluções, não apenas críticas
- **Progressivo**: Sugira próximos passos baseados no nível atual

PRECISÃO NA CLASSIFICAÇÃO
- **Nível do aluno**: Base a classificação no código mais complexo executado
- **Tópicos**: Identifique apenas tópicos realmente demonstrados no código
- **Linguagem**: Seja preciso (ex: "Python 3.x" vs apenas "Python")

```
## VALIDAÇÕES OBRIGATÓRIAS
Antes de finalizar a análise, verifique:
- [ ] Todos os erros foram categorizados corretamente
- [ ] O nível do aluno reflete a complexidade demonstrada
- [ ] Os tópicos identificados estão presentes no código
- [ ] O feedback é acionável e específico
- [ ] A saída JSON está válida e completa

## Entradas
<MENSAGEM_DO_ALUNO>
{{student_message}}
</MENSAGEM_DO_ALUNO>

<CODIGO_DO_ALUNO>
{{student_code}}
</CODIGO_DO_ALUNO>
```

A.1.2 Prompt do Agente Tutor

```
# PERSONA E CONTEXTO

Você é um **Professor de Programação Experiente**, especializado em pedagogia
aplicada ao ensino de desenvolvimento de software. Sua missão é transformar
análises técnicas em experiências de aprendizado profundas e construtivas,
utilizando técnicas educacionais avançadas para desenvolver o
pensamento crítico e autonomia intelectual dos estudantes.

## DIRETRIZ PRINCIPAL
**Maximizar o desenvolvimento do raciocínio autônomo do aluno através
do questionamento orientado, evitando dar respostas prontas e promovendo
a descoberta guiada dos conceitos e soluções.**

# ENTRADA ESPERADA
Você receberá um objeto ‘AnaliseCodigoCompleta‘ contendo:
- Análise geral (nível do aluno, linguagem, objetivos, pontuação)
- Diagnóstico detalhado de erros por categoria
- Feedback construtivo estruturado
- Código melhorado (quando disponível)

# PROCESSO DE PENSAMENTO PEDAGÓGICO
```

```
## 1. DIAGNÓSTICO PEDAGÓGICO
Antes de formular sua resposta, analise sistematicamente:

### Perfil do Estudante
- **Nível de Conhecimento**: Adapte a complexidade da linguagem ao `nivel_aluno`
  - INICIANTE: Foque em conceitos fundamentais, use analogias simples
  - INTERMEDIARIO: Introduza conceitos mais abstratos, questione padrões
  - AVANCADO: Desafie com questões arquiteturais e otimizações
  - EXPERT: Explore trade-offs, design patterns e impactos sistêmicos

### Priorização Educacional
- **Erros de Alto Impacto**: Priorize erros marcados como
`impacto_aprendizado: 'alto'`
- **Conceitos Fundamentais**: Identifique gaps conceituais
nos `topicos_envolvidos`
- **Padrões Recorrentes**: Busque temas comuns entre diferentes
categorias de erro

## 2. ESTRATÉGIA SOCRÁTICA ESTRUTURADA

### Sequência de Questionamento Obrigatória:
1. **Questionamento Diagnóstico**:
"O que você esperava que acontecesse quando...?"
2. **Análise de Causa**:
"Por que você acha que esse comportamento está ocorrendo?"
3. **Exploração de Alternativas**:
"Que outras abordagens você consegue imaginar para..."
4. **Conexão Conceitual**:
"Como isso se relaciona com [conceito fundamental]?"
5. **Aplicação Prática**:
"Em que outras situações você aplicaria esse princípio?"


## 3. ESTRUTURA DE RESPOSTA OBRIGATÓRIA

### Seção 1: RECONHECIMENTO E CONTEXTO
- Reconheça os pontos fortes identificados
- Contextualizar o desafio no nível apropriado
```

```
- Estabeleça conexão empática com a dificuldade

**Template Adaptativo:**
‘‘‘
INICIANTE: "Vejo que você está explorando [conceito].
É normal que [desafio específico] seja confuso no início..."
INTERMEDIARIO: "Percebi que você domina [conceitos básicos]
e agora está enfrentando [conceito mais complexo]..."
AVANCADO: "Seu código demonstra compreensão sólida de [conceitos],
mas vamos explorar algumas nuances..."
EXPERT: "Sua implementação é tecnicamente correta, mas vamos analisar
algumas considerações arquiteturais..."
‘‘‘

### Seção 2: EXPLORAÇÃO GUIADA DE ERROS
Para cada erro de impacto médio/alto, siga a sequência:

1. **Descoberta do Problema**
   - Use perguntas que levem o aluno a identificar o erro
   - "O que você observa quando executa a linha X?"
   - "Como você testaria se esta função está funcionando corretamente?"

2. **Compreensão da Causa**
   - Questione o raciocínio por trás da implementação
   - "Por que você escolheu esta abordagem?"
   - "O que você esperava que acontecesse aqui?"

3. **Exploração de Soluções**
   - Não forneça a correção direta
   - "Que modificações você tentaria para resolver isso?"
   - "Como [conceito relacionado] poderia ajudar nesta situação?"

### Seção 3: DESENVOLVIMENTO CONCEITUAL
- Conecte os erros específicos a conceitos fundamentais
- Use analogias apropriadas ao nível do aluno
- Introduza novos conceitos através de questionamento

**Banco de Analogias por Nível:**
- **INICIANTE**: Analogias do mundo físico
```

```
(receitas, instruções, objetos cotidianos)
- **INTERMEDIARIO**: Analogias organizacionais
(escritórios, protocolos, sistemas)
- **AVANCADO**: Analogias arquiteturais e de engenharia
- **EXPERT**: Analogias sistêmicas e filosóficas

### Seção 4: PRÁTICAS DE MELHORIA ORIENTADAS
Em vez de listar práticas, conduza o aluno à descoberta:

- "Como você tornaria este código mais fácil de entender
para alguém que nunca o viu?"
- "Que padrões você observa no código de desenvolvedores experientes?"
- "Como você organizaria este código se ele fosse crescer 10 vezes de tamanho?"

### Seção 5: DESAFIOS DE REFLEXÃO PROGRESSIVA
Baseado no ‘nivel_aluno‘, proponha desafios graduais:

**INICIANTE:**
- Exercícios de trace manual
- Modificações pequenas e incrementais
- Testes de hipóteses simples

**INTERMEDIARIO:**
- Refatorações guiadas
- Implementação de variações
- Análise de casos limite

**AVANCADO:**
- Design de soluções alternativas
- Análise de trade-offs
- Otimizações orientadas

**EXPERT:**
- Análise de padrões emergentes
- Considerações arquiteturais
- Impactos sistêmicos

# DIRETRIZES DE COMUNICAÇÃO
```

```
## Tom e Linguagem
- **Empático e Encorajador**: Sempre reconheça o esforço e progress
- **Curioso e Investigativo**: Use "pergunto me..."
"será que..." "como você vê..."
- **Paciente e Progressivo**: Construa compreensão camada por camada
- **Específico e Concreto**: Evite generalidades, referencie partes
específicas do código

## Restrições Rígidas
- **NUNCA** forneça código corrigido diretamente
- **NUNCA** liste soluções prontas
- **SEMPRE** termine seções com perguntas reflexivas
- **SEMPRE** conecte erros específicos a princípios gerais
- **SEMPRE** adapte a complexidade ao nível identificado

# VALIDAÇÃO FINAL
Antes de entregar sua resposta, verifique:
- [ ] Adaptei a linguagem ao nível do aluno?
- [ ] Evitei dar soluções prontas?
- [ ] Cada seção termina com questionamento?
- [ ] Conectei erros específicos a conceitos gerais?
- [ ] Inclui desafios progressivos apropriados?
- [ ] O tom é encorajador e construtivo?

## Entradas
<MENSAGEM_DO_ALUNO>
{{student_message}}
</MENSAGEM_DO_ALUNO>

<CODIGO_DO_ALUNO>
{{student_code}}
</CODIGO_DO_ALUNO>

<ANALISE_DO_SENIOR>
{{code_analysis}}
</ANALISE_DO_SENIOR>
```

A.1.3 Exemplo de Análise de Código produzida pelo Agente de Diagnóstico de Códigos


```
{
   "analise_geral":{
      "nivel_aluno":"AVANCADO",
      "linguagem_programacao":"Python 3.x",
      "objetivo_codigo":"O c\u00f3digo implementa opera\u00e7\u00f5es CRUD
      (Criar, Ler, Atualizar, Deletar) para o modelo de paciente,
      incluindo a gest\u00e3o de informa\u00e7\u00f5es
      m\u00e9dicas associadas, como alergias e condi\u00e7\u00f5es
      cr\u00f4nicas. Utiliza SQLAlchemy para intera\u00e7\u00e3o
      com o banco de dados, Pydantic para valida\u00e7\u00e3o de
      dados e ‘asyncio‘ para opera\u00e7\u00f5es ass\u00edncronas.
      H\u00e1 tamb\u00e9m fun\u00e7\u00f5es para gerenciar documentos
      de pacientes em armazenamento na nuvem (GCP) e limpar
      dados de pacientes.",
      "topicos_envolvidos":[
         "Programa\u00e7\u00e3o Ass\u00edncrona",
         "Bancos de Dados (SQLAlchemy ORM)",
         "Modelagem de Dados (Pydantic)",
         "Programa\u00e7\u00e3o Orientada a Objetos",
         "Tratamento de Exce\u00e7\u00f5es",
         "Tipagem Est\u00e1tica",
         "Cloud Computing (GCP)",
         "Estruturas de Dados",
         "Modulariza\u00e7\u00e3o"
      ],
      "pontuacao_geral":75
   },
   "diagnostico_erros":{
      "total_erros":12,
      "erros_por_categoria":{
         "sintaticos":[
            {
               "tipo":"Importa\u00e7\u00e3o Ausente",
               "descricao":"As classes e fun\u00e7\u00f5es ’Session’,
               ’or_’, ’col’, ’delete’ e ’update’ n\u00e3o foram
               importadas do pacote ’sqlalchemy’, o que impede a
               execu\u00e7\u00e3o do c\u00f3digo.",
```

```
            "correcao_sugerida": "Adicione as importa\u00e7\u00f5es
            necess\u00e1rias no in\u00edcio do arquivo:
            ‘from sqlalchemy import select, or_, col, delete,
            update‘ e ‘from sqlalchemy.ext.asyncio import
            AsyncSession as Session‘
            (assumindo AsyncSession para consist\u00eancia).",
            "impacto_aprendizado":"alto"
         }
      ],
      "logicos":[
         {
            "tipo":"Uso Incorreto de Assincronicidade",
            "descricao":"As fun\u00e7\u00f5es ass\u00edncronas
            (‘async def‘) est\u00e3o chamando m\u00e9todos de
            sess\u00e3o do SQLAlchemy de forma s\u00edncrona (‘db.exec‘,
            ‘db.get‘, ‘db.delete‘, ‘db.commit‘, ‘db.refresh‘) em vez
            de aguardar a conclus\u00e3o com ‘await‘. Isso pode causar
            bloqueios e comportamento inesperado em um ambiente
            ass\u00edncrono.",
            "correcao_sugerida":"Use ‘await‘ antes de todas as
            opera\u00e7\u00f5es de banco de dados que interagem com uma
            sess\u00e3o ass\u00edncrona, por exemplo: ‘await
            db.execute(statement)‘, ‘await db.get(Patient, patient_id)‘,
            ‘await db.delete(db_patient)‘, ‘await db.commit()‘,
            ‘await db.refresh(db_patient)‘.",
            "impacto_aprendizado":"alto"
         },
         {
            "tipo":"Inconsist\u00eancia de Tipo de ID",
            "descricao":"O ID do paciente (‘patient_id‘) \u00e9 tratado
            como ‘uuid.UUID‘ em algumas fun\u00e7\u00f5es (e nos modelos
            Pydantic) e como ‘int‘ em outras (‘update_patient‘,
            ‘update_patient_by_field‘, ‘delete_patient‘). Esta
            inconsist\u00eancia pode levar a erros de tipo e falhas
            de busca no banco de dados.",
            "correcao_sugerida":"Padronize o tipo de ‘patient_id‘ para
            ‘uuid.UUID‘ em todas as fun\u00e7\u00f5es onde ele representa
            o identificador \u00fanico do paciente.",
            "impacto_aprendizado":"medio"
```

```
        },
        {
            "tipo":"Manipula\u00e7\u00e3o de Resultado de Consulta",
            "descricao":"Na fun\u00e7\u00e3o ‘get_patients‘, ‘result.all()‘
            pode n\u00e3o retornar objetos ORM esperados ap\u00f3s ‘await
            db.execute()‘. O m\u00e9todo correto para recuperar objetos
            ORM de uma consulta ass\u00edncrona \u00e9
            ‘result.scalars().all()‘.",
            "correcao_sugerida":"Altere ‘result = await
            db.execute(statement)‘ e ‘patients = result.all()‘ para
            ‘result = await db.execute(statement)‘ e
            ‘patients = result.scalars().all()‘.",
            "impacto_aprendizado":"medio"
        },
        {
            "tipo":"Fun\u00e7\u00e3o S\u00edncrona em Contexto
            Ass\u00edncrono",
            "descricao":"A fun\u00e7\u00e3o ‘get_patient_by_username_field‘
            \u00e9 definida como s\u00edncrona, mas suas chamadas de
            ORM (‘session.exec‘) implicam uma intera\u00e7\u00e3o com
            o banco de dados que, no contexto de outras
            fun\u00e7\u00f5es ass\u00edncronas do m\u00f3dulo,
            \u00e9 prov\u00e1vel que precise ser ass\u00edncrona.
            Isso gera inconsist\u00eancia e poss\u00edveis problemas
            de concorr\u00eancia ou bloqueio.",
            "correcao_sugerida":"Transforme a fun\u00e7\u00e3o
            ‘get_patient_by_username_field‘ em uma fun\u00e7\u00e3o
            ss\u00edncrona (‘async def‘) e utilize ‘await
            session.execute(statement)‘ e ‘result.scalar_one_or_none()‘
            para buscar os resultados.",
            "impacto_aprendizado":"medio"
        }
    ],
    "estruturais":[
        {
            "tipo":"Inconsist\u00eancia de Nomenclatura de Par\u00e2metros",
            "descricao":"H\u00e1 inconsist\u00eancia nos nomes dos
            par\u00e2metros da sess\u00e3o do banco de dados, usando
            ora ‘db: Session‘ ora ‘session: Session‘. Embora funcional,
```

```
            a padroniza\u00e7\u00e3o melhora a legibilidade e a
            manuten\u00e7\u00e3o do c\u00f3digo.",
            "correcao_sugerida":"Escolha um nome padr\u00e3o, como
            ‘session‘, e utilize-o consistentemente em todas as
            fun\u00e7\u00f5es que recebem a sess\u00e3o do banco de
            dados como par\u00e2metro.",
            "impacto_aprendizado":"baixo"
        },
        {
            "tipo":"Acoplamento Indireto de L\u00f3gica
            Ass\u00edncrona/S\u00edncrona",
            "descricao":"Embora a maioria das opera\u00e7\u00f5es de
            CRUD sejam ass\u00edncronas, fun\u00e7\u00f5es auxiliares
            como ‘get_patient_document_photo_from_bucket‘ s\u00e3o
            s\u00edncronas. Embora n\u00e3o seja um erro direto,
            misturar chamadas de banco de dados/I/O ass\u00edncronas
            e s\u00edncronas no mesmo m\u00f3dulo pode dificultar
            o racioc\u00ednio sobre o fluxo de execu\u00e7\u00e3o
            e a gest\u00e3o de recursos.",
            "correcao_sugerida":"Avalie se as fun\u00e7\u00f5es
            s\u00edncronas que envolvem I/O (como ‘download_file‘)
            deveriam ser ass\u00edncronas para manter a consist\u00eancia
            do paradigma ass\u00edncrono ou se o escopo dessas
            fun\u00e7\u00f5es deve ser explicitamente gerenciado
            como s\u00edncrono para evitar bloqueios.",
            "impacto_aprendizado":"baixo"
        }
    ],
    "performance":[
        {
            "tipo":"Commits de Transa\u00e7\u00e3o em Loop",
            "descricao":"Na fun\u00e7\u00e3o ‘clear_patient_medical_data‘,
            a chamada ‘session.commit()‘ \u00e9 realizada dentro do loop
            que itera sobre os modelos m\u00e9dicos. Isso resulta em
            m\u00faltiplas transa\u00e7\u00f5es separadas para cada
            exclus\u00e3o, o que \u00e9 altamente ineficiente em
            termos de desempenho de banco de dados.",
            "correcao_sugerida":"Realize o ‘await session.commit()‘
            apenas uma vez, ap\u00f3s todas as opera\u00e7\u00f5es
```

```
            de exclus\u00e3o terem sido executadas no loop. Isso
            consolida as opera\u00e7\u00f5es em uma \u00fanica
            transa\u00e7\u00e3o.",
            "impacto_aprendizado":"alto"
        },
        {
            "tipo":"Sub-otimiza\u00e7\u00e3o de Opera\u00e7\u00f5es
            em Lote",
            "descricao":"A abordagem de deletar modelos m\u00e9dicos
            um por um dentro de um loop, mesmo com um \u00fanico commit
            no final, pode ser otimizada. SQLAlchemy permite
            opera\u00e7\u00f5es em lote mais eficientes.",
            "correcao_sugerida":"Considere o uso de `session.bulk_delete()`
            ou construa uma \u00fanica instru\u00e7\u00e3o `delete` com
            uma cl\u00e1usula `IN` ou `OR` para deletar m\u00faltiplos
            registros de um mesmo modelo em uma \u00fanica
            opera\u00e7\u00e3o de banco de dados.",
            "impacto_aprendizado":"medio"
        }
    ],
    "boas_praticas":[
        {
            "tipo":"Tratamento Gen\u00e9rico de Erros",
            "descricao":"A utiliza\u00e7\u00e3o de `ValueError` para
            indicar que um paciente n\u00e3o foi encontrado \u00e9
            funcional, mas em APIs ou sistemas maiores,
            exce\u00e7\u00f5es mais espec\u00edficas (ou exce\u00e7\u00f5es
            HTTP em frameworks como FastAPI) oferecem um tratamento de
            erros mais claro e granular para o cliente ou para o
            c\u00f3digo chamador.",
            "correcao_sugerida":"Crie exce\u00e7\u00f5es personalizadas
            (e.g., `PatientNotFoundException`) ou utilize as
            exce\u00e7\u00f5es fornecidas pelo framework (e.g.,
            `HTTPException` do FastAPI) para casos de erro
            espec\u00edficos.",
            "impacto_aprendizado":"medio"
        },
        {
            "tipo":"Depend\u00eancia Direta de Utilit\u00e1rios em
```

```
            Fun\u00e7\u00f5es de Limpeza",
            "descricao":"As fun\u00e7\u00f5es ‘clear_patient_medical_data‘
            e ‘clear_patient_basic_data‘ dependem diretamente de
            ‘get_patient_medical_info_models‘ e
            ‘get_patient_basic_data_fields_name‘. Embora seja uma
            modulariza\u00e7\u00e3o, a passagem direta de ‘session‘ e
            ‘patient_id‘/‘phone_number‘ e a re-busca do paciente dentro
            de ‘clear_patient_medical_data‘ quando ‘phone_number‘ \u00e9
            fornecido, poderiam ser encapsulados em uma camada de
            servi\u00e7o mais robusta.",
            "correcao_sugerida":"Considere refatorar a l\u00f3gica de
            busca e limpeza em uma classe de ’servi\u00e7o’ ou
            ’reposit\u00f3rio’ que encapsule as depend\u00eancias
            e o fluxo de trabalho, tornando as fun\u00e7\u00f5es
            mais coesas e menos acopladas.",
            "impacto_aprendizado":"baixo"
         },
         {
            "tipo":"Retorno de Fun\u00e7\u00e3o de Limpeza",
            "descricao":"As fun\u00e7\u00f5es ‘clear_patient_medical_data‘
            e ‘clear_patient_basic_data‘ retornam ‘None‘ e
            ‘result.rowcount‘ (um ‘int‘), respectivamente. A falta
            de um tipo de retorno consistente pode dificultar a
            interface com outras partes do c\u00f3digo que dependem
            de um status claro da opera\u00e7\u00e3o.",
            "correcao_sugerida":"Padronize o retorno das fun\u00e7\u00f5es
            de limpeza para algo mais informativo, como um ‘bool‘
            indicando sucesso/falha, ou um dicion\u00e1rio com um
            resumo da opera\u00e7\u00e3o (linhas afetadas, etc.).",
            "impacto_aprendizado":"baixo"
         }
      ]
   }
},
"feedback_construtivo":{
   "pontos_fortes":[
      "O aluno demonstra um excelente dom\u00ednio da tipagem
      est\u00e1tica (Type Hinting), o que torna o c\u00f3digo mais
      robusto e f\u00e1cil de manter.",
```

```
        "A utiliza\u00e7\u00e3o de Pydantic para valida\u00e7\u00e3o
        e serializa\u00e7\u00e3o de dados (‘PatientCreate‘,
        ‘PatientUpdate‘, ‘model_validate‘, ‘model_dump‘) \u00e9
        uma pr\u00e1tica moderna e eficiente, crucial para o
        desenvolvimento de APIs.",
        "A implementa\u00e7\u00e3o de opera\u00e7\u00f5es ass\u00edncronas
        (‘async def‘) para o CRUD indica um bom entendimento de
        programa\u00e7\u00e3o concorrente e otimiza\u00e7\u00e3o para
        I/O em aplica\u00e7\u00f5es modernas.",
        "O uso do SQLAlchemy ORM com ‘sqlalchemy.future‘
        (assumindo AsyncSession) mostra familiaridade com ferramentas de
        ORM avan\u00e7adas e a preocupa\u00e7\u00e3o com a
        abstra\u00e7\u00e3o do banco de dados.",
        "A estrutura modular do c\u00f3digo, com importa\u00e7\u00f5es bem
        definidas para modelos, configura\u00e7\u00f5es e utilit\u00e1rios,
        \u00e9 um ponto muito positivo.",
        "As docstrings presentes em muitas fun\u00e7\u00f5es s\u00e3o
        claras e descrevem bem o prop\u00f3sito de cada uma."
    ],
    "areas_melhoria":[
        "Aprimorar o gerenciamento de transa\u00e7\u00f5es do SQLAlchemy,
        garantindo que ‘await session.commit()‘ seja usado de forma
        estrat\u00e9gica e eficiente.",
        "Corrigir as inconsist\u00eancias no uso de ‘await‘ para todas as
        opera\u00e7\u00f5es de banco de dados dentro de fun\u00e7\u00f5es
        ass\u00edncronas.",
        "Padronizar o tipo de identificadores de paciente (‘patient_id‘)
        para evitar confus\u00e3o entre ‘uuid.UUID‘ e ‘int‘.",
        "Melhorar a gest\u00e3o de importa\u00e7\u00f5es, assegurando que
        todas as classes e fun\u00e7\u00f5es utilizadas do SQLAlchemy
        estejam devidamente importadas.",
        "Refinar o tratamento de exce\u00e7\u00f5es, usando tipos mais
        espec\u00edficos ou exce\u00e7\u00f5es HTTP, conforme a arquitetura
        da aplica\u00e7\u00e3o."
    ],
    "proximos_passos":[
        "Aprofundar os conhecimentos em SQLAlchemy AsyncSession: explore o uso
        de ‘await session.begin()‘ e ‘await session.rollback()‘ para controle
        expl\u00edcito de transa\u00e7\u00f5es.",
```

```
            "Estudar a fundo ‘sqlalchemy.orm.query.Query.coro_all()‘ e
            ‘sqlalchemy.ext.asyncio.AsyncResult‘ para entender melhor como
            manipular resultados de queries ass\u00edncronas.",
            "Pesquisar sobre Dependency Injection (Inje\u00e7\u00e3o de
            Depend\u00eancia) em frameworks ass\u00edncronos (como FastAPI)
            para gerenciar a sess\u00e3o do banco de dados de forma mais elegante
            e test\u00e1vel.",
            "Implementar exce\u00e7\u00f5es personalizadas para cen\u00e1rios
            espec\u00edficos, como ’Recurso N\u00e3o Encontrado’, para tornar
            a API mais robusta e amig\u00e1vel ao desenvolvedor.",
            "Explorar ‘session.bulk_insert_mappings()‘,
            ‘session.bulk_update_mappings()‘ e ‘session.bulk_delete()‘
            para otimizar opera\u00e7\u00f5es em lote no banco de dados."
         ],
         "recursos_recomendados":[
            "Documenta\u00e7\u00e3o oficial do SQLAlchemy: Se\u00e7\u00e3o
            ’Asynchronous I/O’ e ’Working with ORM Sessions’.",
            "FastAPI e SQLAlchemy Async: Tutoriais e exemplos sobre como
            integrar esses dois em projetos.",
            "Pydantic docs: ’Model methods’ para entender ‘model_validate‘
            e ‘model_dump‘ em profundidade.",
            "PEP 8 - Style Guide for Python Code: Para aprimorar as boas
            pr\u00e1ticas de c\u00f3digo.",
            "Artigos sobre Transaction Management Patterns em Python ORMs."
         ]
      },
      "codigo_melhorado":{
         "incluir":false,
      }
}
```


A.2 Códigos Utilizados para Cenários de Teste

A.2.1 Código Nível Iniciante

```
#include <iostream>
#include <vector>
using namespace std;
```

```
// Problema: Calcular a soma dos elementos de um array
// e contar quantos são maiores que a média
// Input: n (tamanho do array), seguido de n números

int main() {
    int n;
    cout << "Digite o tamanho do array: ";
    cin >> n;

    vector<int> arr(n);
    int soma = 0;

    cout << "Digite os " << n << " números:" << endl;

    for (int i = 0; i <= n; i++) {
        cin >> arr[i];
        soma += arr[i];
    }

    int media = soma / n;

    cout << "Soma total: " << soma << endl;
    cout << "Média: " << media << endl;

    int contador = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] > media) {
            contador++;
        }
    }

    cout << "Números maiores que a média: " << contador << endl;

    cout << "Percentual acima da média: " << (contador * 100) / n
    << "%" << endl;

    cout << "Os números maiores que a média são: ";
    for (int i = 0; i < n; i++) {
        if (arr[i] > media) {
```

```
            cout << arr[i] << " ";
        }
    }
    cout << endl;
    return 0;
}
```

A.2.2 Código Nível Médio

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

const int INF = INT_MAX;

// Problema: Encontrar o menor caminho entre dois vértices
// Input: n vértices, m arestas, origem s, destino t
// Cada aresta tem peso w

struct Edge {
    int to, weight;
};

vector<int> dijkstra(vector<vector<Edge>>& graph, int start, int end) {
    int n = graph.size();
    vector<int> dist(n, INF);
    vector<int> parent(n, -1);
    priority_queue<pair<int, int>> pq; // {distancia, vertice}

    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        int u = pq.top().second;
        int d = pq.top().first;
        pq.pop();

        for (Edge& edge : graph[u]) {
```

```
            int v = edge.to;
            int w = edge.weight;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                parent[v] = u;
                pq.push({dist[v], v});
            }
        }
    }

    vector<int> path;
    int current = end;
    while (current != -1) {
        path.push_back(current);
        current = parent[current];
    }

    reverse(path.begin(), path.end());
    return path;
}

int main() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    vector<vector<Edge>> graph(n);

    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
    }

    vector<int> path = dijkstra(graph, s, t);

    cout << "Caminho: ";
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];
```

```
        if (i < path.size() - 1) cout << " -> ";
    }
    cout << endl;

    return 0;
}
```

A.2.3 Código Nível Avançado

```
import uuid
from typing import Any, List, Optional

from app.models.user import (
    User,
    UserBase,
    UserCreate,
    UserUpdate,
)
from sqlalchemy.future import select

from app.core.config import settings
from app.utils import current_datetime, download_file


def get_user_by_username_field(
    credential_username: str, database_session: Session
) -> User | None:
    query_statement = select(User).where(
        or_(
            col(User.cpf) == credential_username,
            col(User.email) == credential_username,
            col(User.phone_number) == credential_username,
        )
    )
    query_result = database_session.exec(query_statement).first()
    return query_result


# --- CRUD Usuario ---
```

```
async def create_user(
user_data: UserCreate, database_connection: Session
) -> User:
    """Cria um novo usuario no banco de dados."""
    new_user = User.model_validate(
        user_data
    )  # Valida os dados com o modelo
    database_connection.add(new_user)
    database_connection.commit()
    database_connection.refresh(new_user)
    return new_user


async def get_user_by_id(
user_identifier: uuid.UUID, database_connection: Session
) -> User:
    """Busca um usuario pelo ID."""
    query_statement = select(User).where(col(User.id) == user_identifier)
    query_result = database_connection.exec(query_statement)
    user_record = query_result.scalar_one_or_none()
    # if not user_record:
    #     raise ValueError('User Not Found')
    return user_record


async def get_user_by_phone_number(
    contact_phone_number: str, database_connection: Session
) -> User:
    """Busca um usuario pelo número de telefone."""
    query_statement = select(User).where(
        col(User.phone_number) == contact_phone_number
    )
    query_result = database_connection.exec(query_statement)
    user_record = query_result.scalar_one_or_none()
    return user_record


async def get_users(
```

```
    database_connection: Session,
    records_skip: int = 0,
    records_limit: int = 100,
) -> List[User]:
    """Busca uma lista de usuarios."""
    query_statement = select(User).offset(records_skip).limit(records_limit)
    query_result = database_connection.exec(query_statement)
    users_list = query_result.all()
    return list(users_list)


async def update_user(
    user_identifier: int,
    user_update_data: UserUpdate,
    database_connection: Session,
) -> Optional[User]:
    """Atualiza os dados de um usuario."""
    existing_user = await database_connection.get(User, user_identifier)
    if not existing_user:
        raise ValueError('User Not Found')

    # Pega os dados do Pydantic model que não são None
    update_fields_data = user_update_data.model_dump(exclude_unset=True)
    for field_key, field_value in update_fields_data.items():
        setattr(existing_user, field_key, field_value)

    # Atualiza o timestamp de atualização (se existir no modelo)
    if hasattr(existing_user, 'updated_at'):
        existing_user.updated_at = current_datetime()

    database_connection.add(existing_user)
    database_connection.commit()
    database_connection.refresh(existing_user)
    return existing_user


async def update_user_by_field(
    user_identifier: int,
    target_field: str,
```

```
    field_value: Any,
    database_connection: Session,
) -> Optional['User']:
    """Atualiza os dados de um usuario por um Campo."""
    permitted_fields = list(UserBase.model_fields.keys())
    if target_field not in permitted_fields:
        raise ValueError(
            f'Invalid field ({target_field}).'
        )

    user_record = await database_connection.get(User, user_identifier)
    if not user_record:
        raise ValueError('User Not Found')

    setattr(user_record, target_field, field_value)

    database_connection.add(user_record)
    database_connection.commit()
    database_connection.refresh(user_record)
    return user_record


async def delete_user(
user_identifier: int, database_connection: Session
) -> bool:
    """Remove um usuario do banco de dados."""
    existing_user = await database_connection.get(User, user_identifier)
    if not existing_user:
        return False
    database_connection.delete(existing_user)
    database_connection.commit()
    return True


def get_user_document_photo_from_bucket(
    document_storage_object_identifier: str,
):
    if not document_storage_object_identifier:
        raise ValueError(
```

```
            'User Does not have a Document With Photo Persisted'
        )
    downloaded_file_path = download_file(
        bucket_name=settings.USER_DOCUMENT_BUCKET_NAME,
        object_name=document_storage_object_identifier,
    )
    return downloaded_file_path


def get_user_basic_data_fields_name(include_authentication_storage_column=True):
    user_basic_data_field_names = [
        field_name for field_name, model_field in UserBase.model_fields.items()
        if field_name != 'phone_number'
    ]
    if include_authentication_storage_column:
        user_basic_data_field_names.append(
            'face_document_storage_object_name'
        )
    return user_basic_data_field_names


def clear_user_basic_data(
    *,
    database_session: Session,
    user_identifier: uuid.UUID | None = None,
    contact_phone_number: str = None,
    remove_all_users: bool = False,
):
    data_field_names = get_user_basic_data_fields_name()
    clearing_update_data = {data_field: None for data_field in data_field_names}

    if user_identifier:
        update_statement = (
            update(User)
            .where(col(User.id) == user_identifier)
            .values(**clearing_update_data)
        )
    elif contact_phone_number:
        update_statement = (
```

```
            update(User)
            .where(col(User.phone_number) == contact_phone_number)
            .values(**clearing_update_data)
        )
    elif remove_all_users:
        update_statement = update(User).values(**clearing_update_data)
    else:
        raise ValueError(
            'Deve especificar user_id, phone_number ou remove_all_users=True'
        )

    execution_result = database_session.exec(update_statement)
    database_session.commit()

    return execution_result.rowcount
```

A.2.4 Código Nível Expert

```
import asyncio
import functools
import oci
import re
import uuid
import psycopg
import src.constants as const
import src.utils
import time
import typing as types

from app.core.settings import app_settings
from app.api.deps import DatabaseSession
from app.models.rag_metadata import DocumentMetadata
from contextlib import contextmanager
from dotenv import load_dotenv
from langchain_community.chat_message_histories import (
    FileChatMessageHistory,
)
from langchain_community.chat_models import ChatOCIGenAI
from langchain_community.embeddings import OCIGenAIEmbeddings
```

```
from langchain_core.callbacks import BaseCallbackHandler
from langchain_core.documents import Document
from langchain_core.messages import BaseMessage
from langchain_core.prompts import (
    ChatPromptTemplate,
    MessagesPlaceholder,
)
from langchain_core.runnables import (
    RunnableSerializable,
    RunnableConfig,
)
from langchain_postgres import PostgresChatMessageHistory
from langchain_qdrant import Qdrant
from qdrant_client import qdrant_client
from pathlib import Path
from pydantic import BaseModel
from sqlmodel import select, Field
from src.logger import log
from typing import Any, Callable, TypedDict, Union, Optional, List, Dict
from uuid import UUID


load_dotenv()


#
# def timeit(func):
#     @wraps(func)
#     def wrapper(*args, **kwargs):
#         import time
#
#         start = time.time()
#         result = func(*args, **kwargs)
#         end = time.time()
#
#         execution_time = end - start
#         truncated_time = round(execution_time, 2)
#         log.info(f'{func.__name__} took {truncated_time} seconds to execute')
#
```

```
#         # detailed logging
#         log.debug(f'args: {args}')
#         log.debug(f'kwargs: {kwargs}')
#         log.debug(f'result: {result}')
#
#         return result
#
#     return wrapper


def timeit(function_to_time):
    @contextmanager
    def timing_context():
        initial_timestamp = time.time()
        yield
        elapsed_duration = time.time() - initial_timestamp
        rounded_duration = round(elapsed_duration, 2)


    @functools.wraps(function_to_time)
    def timing_wrapper(*function_args, **function_kwargs):
        # detailed logging
        log.debug(f'args: {function_args}')
        log.debug(f'kwargs: {function_kwargs}')
        if not asyncio.iscoroutinefunction(function_to_time):
            with timing_context():
                function_result = function_to_time(
                *function_args, **function_kwargs
                )
                log.debug(f'result: {function_result}')
                return function_result
        else:

            async def async_wrapper():
                with timing_context():
                    function_result = await function_to_time(
                    *function_args, **function_kwargs
                    )
                    log.debug(f'result: {function_result}')
```

```
                    return function_result

            return async_wrapper()

    return timing_wrapper


def _is_valid_identifier(identifier_value: str) -> bool:
    """Check if the value is a valid identifier."""
    # Use a regular expression to match the allowed characters
    pattern_validator = re.compile(r'^[a-zA-Z0-9-_]+$')
    return bool(pattern_validator.match(identifier_value))


def validate_field_spec(user_session_id: str) -> None:
    """Validate the session id."""
    if not _is_valid_identifier(user_session_id):
        raise ValueError(
            f'Session ID {user_session_id} is not in a valid format. '
            'Session ID must only contain alphanumeric characters, '
            'hyphens, and underscores. Please provide a valid session id '
            'via config.'
        )


def create_session_factory(
    storage_base_dir: Union[str, Path] = 'chat_histories',
    database_enabled: bool = False,
    synchronous_connection: psycopg.Connection = None,
    asynchronous_connection: psycopg.AsyncConnection = None,
) -> Callable[[str], PostgresChatMessageHistory] | Callable[
    [Any], FileChatMessageHistory
]:
    """Create a session factory for chat histories."""
    if not database_enabled:
        storage_directory = Path(storage_base_dir)
        if not storage_directory.exists():
            storage_directory.mkdir(parents=True)
```

```
def get_chat_history_from_file(chat_session_id) -> FileChatMessageHistory:
    """Get a chat history from a user id and conversation id."""
    validate_field_spec(session_id=chat_session_id)
    history_file_path = storage_directory / f'{chat_session_id}.json'
    return FileChatMessageHistory(str(history_file_path))

def get_chat_history_from_postgres(
    chat_session_id: str = str(uuid.uuid4()),
) -> PostgresChatMessageHistory:
    """Get a chat history from a user id and conversation id."""

    validate_field_spec(session_id=chat_session_id)

    if synchronous_connection:
        postgres_chat_instance = PostgresChatMessageHistory(
            const.POSTGRES_MESSAGE_HISTORY_TABLE_NAME,
            chat_session_id,
            sync_connection=synchronous_connection,
        )
        return postgres_chat_instance

    if asynchronous_connection:
        postgres_chat_instance = PostgresChatMessageHistory(
            const.POSTGRES_MESSAGE_HISTORY_TABLE_NAME,
            chat_session_id,
            async_connection=asynchronous_connection,
        )
        return postgres_chat_instance

    raise ValueError(
        'Please give a async/sync connection or change use_db to False'
    )

return (
    get_chat_history_from_postgres
    if database_enabled
    else get_chat_history_from_file
)
```

```
def get_chat_history(chat_session_id: str, history_factory: Callable) -> Any:
    """Get chat history from a session id."""
    return history_factory(chat_session_id)


class LLMStartingGenerationCallback(BaseCallbackHandler):
    """A callback to log the start of the LLM generation."""

    def __init__(self):
        self.start_time = time.time()
        self.end_time = None

    def calc_model_generation_delay(self):
        self.end_time = time.time()
        return self.end_time - self.start_time

    def on_chat_model_start(
        self,
        serialized: Dict[str, Any],
        messages: List[List[BaseMessage]],
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[Dict[str, Any]] = None,
        **kwargs: Any,
    ) -> Any:
        log.info(f'1Starting generation...')
        log.info(
            f'1Model generation delay: {self.calc_model_generation_delay()}s'
        )

    def on_llm_start(
        self,
        serialized: types.Dict[str, Any],
        prompts: List[str],
        *,
        run_id: uuid.UUID,
```

```
        parent_run_id: Optional[uuid.UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[types.Dict[str, Any]] = None,
        **kwargs: Any,
    ) -> Any:
        log.info(f'2Starting generation...')
        log.info(
            f'2Model generation delay: {self.calc_model_generation_delay()}s'
        )
        return None

    def on_chain_start(
        self,
        serialized: Dict[str, Any],
        inputs: Dict[str, Any],
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[Dict[str, Any]] = None,
        **kwargs: Any,
    ) -> Any:
        self.start_time = time.time()
        log.info(f'Starting chain...')
        return None


def get_llm_model(
    **model_kwargs,
):
    """A function to load a LLM ChatModel."""

    if not const.IS_OCI_CREDENTIALS_VALID:
        return

    oci_client = oci.generative_ai_inference.GenerativeAiInferenceClient(
        config=const.OCI_CREDENTIALS,
        service_endpoint=const.DEFAULT_GENAI_SERVICE_ENDPOINT,
    )
```

```
    language_model = ChatOCIGenAI(
        model_id='meta.llama-3.1-70b-instruct',
        service_endpoint=const.DEFAULT_GENAI_SERVICE_ENDPOINT,
        compartment_id=const.DEFAULT_COMPARTMENT_ID,
        model_kwargs={
            'max_tokens': const.MAX_OUTPUT_TOKENS,
            'temperature': 0.4,
        },
        client=oci_client,
        callbacks=[LLMStartingGenerationCallback()],
    )

    return language_model


def get_embedding_model(
    embedding_model_identifier: str = const.DEFAULT_OCI_EMBEDDING_MODEL,
):
    if not const.IS_OCI_CREDENTIALS_VALID:
        return

    oci_client = oci.generative_ai_inference.GenerativeAiInferenceClient(
        const.OCI_CREDENTIALS
    )

    embedding_instance = OCIGenAIEmbeddings(
        model_id=embedding_model_identifier,
        service_endpoint=const.DEFAULT_GENAI_SERVICE_ENDPOINT,
        compartment_id=const.DEFAULT_COMPARTMENT_ID,
        client=oci_client,
    )

    return embedding_instance


def get_vector_index(
    *,
    client: qdrant_client.QdrantClient,
    collection_name: str = const.COLLECTION_NAME,
```

```
    embedded_model: OCIGenAIEmbeddings,
) -> Qdrant:
    qdrant_storage = Qdrant(
        client,
        collection_name=collection_name,
        embeddings=embedded_model,
    )
    return qdrant_storage


def get_retriever(
    *, collection_name, engine=None, q_client=None, embed_model=None
):
    if engine is None:
        vector_index = get_vector_index(
            client=q_client,
            collection_name=collection_name,
            embedded_model=embed_model,
        )
        engine = vector_index

    return engine.as_retriever(
        search_kwargs={'k': 10},
    )


def get_vector_db_client(
    *,
    host: types.Optional[str] = 'localhost',
    port: types.Optional[int] = 6333,
    api_key: types.Optional[str] = None,
):
    qdrant_client_instance = qdrant_client.QdrantClient(
        host=host,
        port=port,
        api_key=api_key,
    )
    return qdrant_client_instance
```

```
class InputChat(TypedDict):
    """Input for the chat endpoint."""

    input: str
    """Human input"""


# User input
class ChatHistory(BaseModel):
    """Chat history with the bot."""

    question: str


embedding_model_instance = get_embedding_model()
vector_database_client = src.rag.get_vector_db_client(**const.VECTOR_DB_KWARGS)


class RRetriever(RunnableSerializable):
    collection_name: str

    def get_relevant_documents(self, user_input: str) -> List[Document]:
        """Get relevant documents."""
        vector_store = Qdrant(
            vector_database_client,
            collection_name=self.collection_name,
            embeddings=embedding_model_instance,
        )
        document_retriever = vector_store.as_retriever(search_kwargs={'k': 10})
        return document_retriever.invoke(user_input)

    def invoke(
        self, user_input: str, config: Optional[RunnableConfig] = None
    ) -> List[Document]:
        """Invoke the retriever."""
        vector_store = Qdrant(
            client,
            collection_name=self.collection_name,
```

```
            embeddings=embedding_model_instance,
        )
        document_retriever = vector_store.as_retriever(search_kwargs={'k': 4})
        return document_retriever.invoke(user_input, config=config)


class Request(BaseModel):
    """Request model for the chat endpoint."""

    input: str = Field()


def get_custom_retriever(
collection_identifier
) -> RunnableSerializable[Any, Any]:
    """Get the custom retriever."""
    return RRetriever(collection_name=collection_identifier)


def get_collection_name_by_prompt_id(
database_session: DatabaseSession, prompt_id: int
):
    query_statement = select(
    DocumentMetadata.collection_name
    ).where(DocumentMetadata.id == prompt_id)
    query_results = database_session.exec(query_statement).one_or_none()
    return query_results


# <Get Prompt By Prompt ID>
def get_prompt_by_prompt_id(document_metadata: DocumentMetadata):
    template_generator = document_metadata.prompt_generator_template
    chat_prompt = ChatPromptTemplate.from_messages(
        [
            ('system', template_generator),
            MessagesPlaceholder('chat_history'),
            ('human', '{input}'),
        ]
    )
```

```
    return chat_prompt


TITLE_GENERATOR_PROMPT = """
Você sabe gerar títulos para prompts.
Gere um título curto e descritivo para o prompt: "{prompt}".
O título deve conter no máximo 10 palavras.
Retorne somente o título gerado. Não responda à pergunta.
"""
```