Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 03/08/2025; 1st round notif.: 14/11/2025; New version: 05/01/2026; 2nd round notif.: 20/02/2026;
Camera ready: 05/03/2026; Edition review: 24/03/2026; Available online: 24/03/2026; Published: 24/03/2026;

Desenvolvimento de Ferramentas Educacionais para Computação com Apoio de Modelos Generativos de Linguagem

Title: Development of AI-Assisted Teaching Tools for Computer Science Education

Título: Creación de Herramientas de Enseñanza de Computación Basadas en Modelos Generativos de Lenguaje

Hugo Costa Araújo
Universidade Federal de Viçosa
ORCID: 0009-0009-6781-8679
hugo.costa@ufv.br

Matheus Otávio Lisboa
Universidade Federal de Viçosa
ORCID: 0009-0008-7689-4297
matheus.o.lisboa@ufv.br

Pedro Henrique Coura Pereira
Universidade Federal de Viçosa
ORCID: 0009-0008-3115-1561
pedro.coura@ufv.br

Isabela de Castro Freitas
Universidade Federal de Viçosa
ORCID: 0009-0004-2686-8825
isabela.castro.freitas@ufv.br

Maria Lúcia Bento Villela
Universidade Federal de Viçosa
ORCID: 0000-0001-6189-1300
maria.villela@ufv.br

Ricardo Ferreira
Universidade Federal de Viçosa
ORCID: 0000-0003-1802-7829
ricardo@ufv.br

Resumo

Este trabalho realiza uma análise quantitativa e qualitativa de modelos de linguagem de grande escala (LLM) para o ensino de computação, avaliando seu desempenho em diversos aspectos educacionais. Inicialmente, avaliamos o desempenho de quatro modelos (ChatGPT, Claude, Copilot e Gemini) para o desenvolvimento de interfaces gráficas em Python no ambiente Google Colab, explorando diversos formatos de requisições e definindo métricas para compará-los. As métricas de avaliação incluem erros de compilação, execução e funcionalidade, além do tamanho da requisição, do número de interações e do número de linhas de código geradas. A partir de mais de 350 ensaios, foram geradas mais de 250 ferramentas funcionais, alcançando uma taxa de sucesso de 84%. Posteriormente, avaliamos a capacidade de tradução de código de Python para JavaScript, explorando os recursos variados para interface gráfica em navegadores e o desempenho das LLMs para JavaScript, no qual incluímos os modelos Gemini-Pro e DeepSeek. Avaliamos também o desempenho em interfaces com recursos de edição para problemas de grafos e criação de linguagens de domínio específico para ensino de grafos e modelos de aprendizado de máquina. Elaboramos também exemplos de outros usos, como geração de questionários, avaliação de questionários e avaliação de código gerado. Os exemplos avaliados abrangem diversos domínios, todos documentados e disponibilizados como um conjunto de dados. Os resultados indicam uma aceleração significativa na criação de ferramentas educacionais interativas e atrativas, destacando direções promissoras para o uso de LLMs no desenvolvimento de recursos didáticos para computação.

Palavras-chave: *Modelos Generativos de Linguagem; Ferramentas de Ensino de Computação; Interfaces Interativas; Educação em Computação.*

Abstract

This work conducts a quantitative and qualitative analysis of large language models (LLMs) for computer science education, evaluating performance across various educational aspects. Initially, we evaluated the performance of four models (ChatGPT, Claude, Copilot, and Gemini) for developing graphical interfaces in Python within the Google Colab environment, exploring diverse request formats and defining metrics for comparison. Evaluation metrics include compilation, execution, and functionality errors, as well as request size, number of interactions, and number

Cite as: Araújo, H. C., Lisboa, M., Pereira, P. H. C., Freitas, I. C., Villela, M. L. B., & Ferreira, R. S. (2026). Desenvolvimento de Ferramentas Educacionais para Computação com Apoio de Modelos Generativos de Linguagem. Revista Brasileira de Informática na Educação, vol. 34, pp. 279–313. https://doi.org/10.5753/rbie.2026.6311.

of generated code lines. From over 350 trials, 240 functional tools were generated, achieving an 84% success rate. Subsequently, we evaluated the capability for translating code from Python to JavaScript, exploring varied resources for graphical interfaces in browsers and LLM performance for JavaScript. We also assessed performance for interfaces with editing capabilities for graph problems and creation of domain-specific languages for teaching graphs and machine learning models. We further developed examples of other applications such as quiz generation, quiz evaluation, and generated code assessment. The evaluated examples span diverse domains, all documented and made available as a dataset. Results indicate significant acceleration in creating interactive and engaging educational tools, highlighting promising directions for using LLMs in developing educational resources for computer science.
Keywords: *arge Language Models; Computer Science Teaching Tools; Interactive Interfaces; Computer Science Education.*

Resumen

Este trabajo realiza un análisis cuantitativo y cualitativo de modelos de lenguaje de gran escala (LLM) para la enseñanza de computación, evaluando el rendimiento en diversos aspectos educativos. Inicialmente, evaluamos el rendimiento de cuatro modelos (ChatGPT, Claude, Copilot, Gemini y DeepSeek) para el desarrollo de interfaces gráficas en Python en el entorno Google Colab, explorando diversos formatos de solicitudes y definiendo métricas para compararlas. Las métricas de evaluación incluyen errores de compilación, ejecución y funcionalidad, además del tamaño de la solicitud, número de interacciones y número de líneas de código generadas. A partir de más de 350 ensayos, se generaron 240 herramientas funcionales, alcanzando una tasa de éxito del 84 %. Posteriormente, evaluamos la capacidad de traducción de código de Python a JavaScript, explorando los recursos variados para interfaces gráficas en navegadores y el rendimiento de las LLMs para JavaScript. También evaluamos el rendimiento para interfaces con recursos de edición para problemas de grafos y creación de lenguajes de dominio específico para la enseñanza de grafos y modelos de aprendizaje automático. Elaboramos también ejemplos de otros usos como generación de cuestionarios, evaluación de cuestionarios y evaluación de código generado. Los ejemplos evaluados abarcan diversos dominios, todos documentados y disponibilizados como un conjunto de datos. Los resultados indican una aceleración significativa en la creación de herramientas educativas interactivas y atractivas, destacando direcciones prometedoras para el uso de LLMs en el desarrollo de recursos didácticos para computación.
Palabras clave: *Modelos Generativos de Lenguaje; Herramientas de Enseñanza de Computación; Interfaces Interactivas; Educación en Computación.*

1 Introdução

O avanço da capacidade dos Modelos de Linguagem de Grande Escala (LLMs – *Large Language Models*) vem possibilitando a criação de diversas ferramentas computacionais. Uma pesquisa recente da Universidade de Elon mostrou que 52% dos adultos americanos utilizam modelos de linguagem de grande escala, como a ChatGPT (Elon University, 2025), demonstrando a rápida adoção dessas tecnologias. Estes modelos têm a capacidade de gerar respostas coerentes e relevantes para questões relacionadas a diferentes contextos, porém apresentam limitações importantes. A qualidade das respostas pode ser influenciada por diversos fatores, incluindo a requisição (ou *prompt*) fornecida ao modelo, os hiperparâmetros utilizados e a diversidade dos dados de treinamento.

O escopo das LLMs no meio educacional se ampliou com os modelos multimodais (texto, som, vídeo, etc.) (Yuan et al., 2025), que podem auxiliar na formulação e resolução de problemas. O uso da metodologia RAG (*Retrieval-Augmented Generation*) tem se mostrado eficaz para guiar estudantes na solução de problemaspor meios da elaboração de uma série de questões (Yang & Zhu, 2024). Outro caminho é o uso da abordagem REACT (*Reasoning and Acting*) (Yao et al., 2022) para a produção de *prompts* que levam as LLMs a gerarem textos estruturados, quebrando problemas complexos em passos intermediários, juntamente com textos de ações que utilizam as LLMs para resolver os passos criados (Yang et al., 2023). Pesquisas recentes demonstram o potencial das LLMs para melhorar planos de ensino através da simulação de interações professor-aluno (Hu et al., 2025), enquanto outros trabalhos exploram o uso de LLMs como agentes educacionais para análise de dados complexos (Chu et al., 2025a). Além disso, ambientes como o Low-code LLM têm sido desenvolvidos para criar interfaces gráficas sobre modelos de linguagem (Cai et al., 2023), e estudos mostram como novatos utilizam geradores de código com LLMs para resolver tarefas de programação em ambientes de aprendizado autodirigido (Kazemitabaar et al., 2023).

Recentemente, ocorreu uma explosão de trabalhos acadêmicos usando as facilidades oferecidas pelas LLMs, como ressalta um estudo (Joel et al., 2024) que mostrou que mais de 27 mil trabalhos foram publicados entre 2020 e 2024 para entender a capacidade de compreensão das LLMs em domínios especializados. Entretanto, ainda existe espaço para a exploração de novas ideias, como, por exemplo, a geração de figuras estruturadas, como diagramas de blocos, que ainda é pouco explorada no domínio das LLMs (Al-Shetairy et al., 2024; Zala et al., 2023).

Trabalhos recentes têm buscado automatizar o processo e estabelecer critérios quantitativos, incluindo a definição de conjuntos de testes, como o SWE-bench, e o emprego de avaliação automatizada (Jimenez et al., 2023). Contudo, ao se incorporar a dimensão de visualização gráfica e de interação com o usuário, a complexidade da tarefa aumenta substancialmente (L. Yan et al., 2024; Yuan et al., 2025). Dessa forma, a área ainda carece de investigações exploratórias, da construção de conjuntos de teste e do desenvolvimento de métricas para avaliação das ferramentas.

Este trabalho tem como objeto de pesquisa a avaliação exploratória de LLMs comerciais para o desenvolvimento de ferramentas visuais e interativas para o ensino de estruturas de dados e algoritmos. As principais questões que buscamos responder são: como documentar o processo exploratório? Qual a efetividade dos *prompts* utilizados? Como mensurar a eficácia das ferramentas no estado atual? E em quais outras direções podemos expandir a avaliação exploratória?

Este artigo é uma extensão do trabalho anterior (Lisboa et al., 2025), no qual avaliamoquatroro LLMs comerciais: ChatGPT, Claude, Copilot e Gemini. Todo o processo exploratório foi documentado, incluindo a formulação de *prompts* e a análise de sua efetividade na geração de ferramentas educacionais gráficas e interativas para o ensino de problemas clássicos de computação. Em função do bom desempenho das LLMs (Joel et al., 2024; Yuan et al., 2025) na geração de código em *Python* e das facilidades oferecidas pelo ambiente Google Colab, os experimentos exploratórios utilizaram esses recursos. A avaliação incluiu métricas qualitativas, uma vez que a validação das interfaces é um processo manual, mas também incorporou métricas quantitativas sempre que possível, como o sucesso na compilação, número de linhas de código, entre outras.

Avançando nessa linha de investigação, o presente estudo amplia a análise exploratória. Primeiro, avaliamos a geração de interfaces mais interativas que incorporem recursos de editores gráficos, como a edição visual de grafos. Segundo, investigamos o uso de Javascript em substituição ao Python. Javascript executa nativamente em navegadores, oferecendo recursos gráficos interativos por meio de APIs como Canvas, SVG e WebGL. No próprio Google Colab, códigos em JavaScript podem ser executados sem que o notebook esteja conectado aos servidores da Google, utilizando apenas os recursos do navegador local. Adotamos duas abordagens: a primeira avalia a qualidade do código gerado por LLMs populares para os mesmos *prompts* utilizados em Python; a segunda examina a capacidade das LLMs de traduzir código Python para JavaScript voltado à construção de ferramentas educacionais.

Estendendo a avaliação exploratória das LLMs em uma terceira direção, analisamos ainda a capacidade de autoavaliação, utilizando códigos gerados por uma LLM e avaliados por outra, com o objetivo de incentivar estudantes a refletirem sobre a qualidade do código produzido para ferramentas gráficas de ensino. Como quarta direção, exploramos a geração de material didático por meio da criação de interpretadores para linguagens de domínio específico (DSL), área em que as LLMs demonstram forte potencial (Joel et al., 2024). Dois exemplos foram desenvolvidos: o primeiro voltado para uma DSL de grafos e o segundo para uma DSL aplicada à inteligência artificial. Finalmente, visando à fixação de conceitos, construímos exemplos adicionais para avaliar a capacidade das LLMs para gerar questionários executáveis no ambiente Google Colab e em interfaces mais restritas, como as de telefones celulares.

Como contribuições, podemos destacar: primeiro, a criação de um conjunto de dados (*dataset*) inicial de ferramentas visuais para problemas clássicos. Existem ainda poucos trabalhos na área de *datasets* para diagramas estruturados (Sato et al., 2024), sendo que nossa abordagem vai além, ao incorporar simulação interativa aos diagramas. Uma segunda contribuição é o uso de métricas quantitativas (tamanho do prompt, linhas de código, ...) e qualitativas (interface funcional) para guiar o desenvolvimento. A terceira contribuição é uma metodologia para avaliação da capacidade das LLMs na criação de código para algoritmos com visualização. Enquanto a avaliação do uso de LLMs para ensino de programação já é bem estudada (Kiesler & Schiffner, 2023), existe uma lacuna na criação de material interativo e visual. Uma vez explorada a geração de interface, estendemos os exemplos para avaliar a capacidade de tradução de um código para outra linguagem, interpretadores de linguagens de domínio específico, geração de questionários, avaliação de código e editores gráficos iterativos.

Este trabalho está estruturado da seguinte forma: a Seção 2 destaca os trabalhos correlatos, a Seção 3 apresenta os fundamentos das técnicas de *prompt*, a Seção 4 introduz a metodologia, a Seção 5 apresenta um resumo das centenas de exemplos que foram gerados com uma apresentação

visual e a Seção 6 mostra uma avaliação qualitativa e as métricas de avaliação quantitativa. Por fim, a Seção 7 apresenta as conclusões e direções futuras.

2 Trabalhos Relacionados

As LLMs vêm sendo avaliadas em praticamente todos os domínios do conhecimento (Ma et al., 2025). Nosso foco é o domínio educacional em ciência da computação, no qual podemos destacar diversas linhas de pesquisa, como o apoio a alunos iniciantes em programação (Kazemitabaar et al., 2023; Mutanga et al., 2025; Pereira Filho et al., 2025), a correção de código em repositórios, como no caso do SWE-bench (Jimenez et al., 2023), e o auxílio ao desenvolvimento de aplicativos para dispositivos móveis (Vasconcelos & Frota, 2025), dentre outras. Neste trabalho, buscamos avaliar não apenas ferramentas educacionais, mas também a geração de software com componentes visuais L. Yan et al., 2024.

Na avaliação de atividades de ensino introdutório de programação, um estudo recente (Pereira Filho et al., 2025) realizou 46 testes com códigos simples e apresentou uma taxa de sucesso entre 80–90%, sendo que a LLM Gemini obteve os piores resultados, variando entre 70–90%. Entretanto, os *prompts* e os códigos utilizados não foram disponibilizados, e os autores destacam que a natureza do estudo ainda é exploratória. Em outro trabalho recente (F. L. B. Martins et al., 2025), as LLMs não se mostraram adequadas para provas de lógica formal. Uma contribuição importante desse estudo foi a disponibilização das 41 questões avaliadas; contudo, foi utilizado apenas um *prompt* simples por questão, e todo o processo de curadoria foi manual.

A maioria dos trabalhos ainda se encontra em uma fase exploratória, como, por exemplo, a geração de imagens com estudantes do ensino médio (Russo et al., 2023) e o auxílio ao desenvolvimento de aplicativos para celulares (Vasconcelos & Frota, 2025). Um estudo recente que analisou mais de 250 trabalhos na área de educação com LLMs propõe uma taxonomia para seu uso no ensino (Chu et al., 2025b), organizada em duas grandes categorias: agentes pedagógicos e agentes de domínio específico. Os agentes pedagógicos incluem subáreas voltadas a professores, como simuladores de cursos, sistemas de *feedback* e recomendação e subáreas voltadas a estudantes, como aprendizado adaptativo e detecção e correção de erros. Na categoria de domínio específico, a taxonomia abrange áreas de ciências (matemática, etc.), linguagens (escrita, ...) e desenvolvimento profissional, incluindo direito e medicina. No contexto educacional, o estudo destaca ferramentas de geração de código: *ToolCoder, SWE-agent, AgentCoder e MapCoder*.

No contexto de geração de software com interface visual, a área ainda conta com poucos trabalhos. Entre eles, destaca-se a criação de um conjunto de testes para geração multimodal, denominado SWE-M (L. Yan et al., 2024), que contém 617 tarefas coletadas de 17 bibliotecas escritas em Javascript para interface web, visualização de dados e manipulação interativa. Os resultados mostram que as LLMs apresentam baixa taxa de acerto e que ainda é necessária intensa iteração manual para filtragem e avaliação das tarefas. Outro ponto relevante diz respeito às percepções docentes e às questões éticas relacionadas ao uso de IA generativa. Um estudo recente (Verhalen et al., 2025) indica que as análises sobre percepções e impactos na prática pedagógica ainda são preliminares e devem evoluir nos próximos anos. Nosso trabalho exploratório diferencia-se dos demais ao apresentar uma avaliação inicial do potencial das LLMs no domínio da construção de ferramentas visuais e interativas para o ensino de algoritmos e estruturas de dados em computação.

3 Fundamentos

O desenvolvimento das requisições ou *prompts* pode seguir algumas diretrizes para facilitar e melhorar as respostas das ferramentas de LLM. As diretivas mais usuais (Logan IV et al., 2021) seguem o seguinte fluxo: **Contexto:** fornece informações para que a IA entenda o cenário ou a situação; **Tom:** define o estilo desejado para a resposta (formal, informal, amigável, técnico, etc.), ajudando a IA a adequar-se ao tipo de comunicação pretendido; **Clareza e Especificidade:** estabelece a importância de ser claro e específico sobre o que se deseja obter, evitando perguntas vagas que podem gerar respostas imprecisas; **Objetivo:** indica a finalidade do *prompt*, como buscar uma explicação, sugestão ou análise; **Exemplos:** fornece modelos do tipo de resposta esperada, auxiliando a LLM na compreensão da tarefa; **Limitações:** especifica restrições que o usuário deseja impor à resposta (como tópicos ou estilos a serem evitados) e o **Formato:** define a estrutura desejada para a saída (lista, programa, parágrafo, tabela, etc.). A seguir, ilustraremos nossa abordagem com um exemplo de *prompt*.

1. **Contexto**: Desenvolver uma ferramenta de ensino do método de Kruskal para árvore geradora usando Google Colab, *Iwidget* para interfaces interativas e a biblioteca Graphviz.
2. **Tom**: Incluir um botão para executar passo a passo e o “reset"para sortear um novo grafo.
3. **Clareza e Especificidade**: Mostrar com Graphviz gerando PNG para visualizar com destaque os vértices e arestas incluso a cada passo.
4. **Objetivo**: Ferramenta para ensino do algoritmo de Kruskal de árvore geradora.
5. **Exemplos**: Gere como exemplos grafos com 10 a 15 vértices.
6. **Limitações**: Sempre que executar um passo, atualizar o desenho.
7. **Formato**: Usar Graphviz para os grafos.

Os estilos mais recomendados para um *prompt* são fornecer instruções claras e precisas (Xu et al., 2023), utilizando delimitadores como aspas para separar as instruções de exemplos ou trechos que se deseja melhorar. As estratégias são classificadas como **one-shot** quando acompanhadas de um exemplo, ou **few-shot** quando utilizam múltiplos exemplos (Logan IV et al., 2021). Por outro lado, são denominadas **zero-shot** quando utilizam um *prompt* simples sem exemplos.

Uma das estratégias mais populares é a “Chain-of-Thought” (CoT) (Wei et al., 2022), na qual a requisição de um problema complexo é decomposta em uma sequência de etapas menores, explicando o raciocínio em cada etapa durante a interação com a LLM. Outra técnica semelhante é a “least-to-most prompting” (Zhou et al., 2022), que consiste em decompor um problema complexo em uma série de subproblemas mais simples, que são então resolvidos sequencialmente. Existe também a “golden chain-of-thought”, que, além de decompor o problema em uma sequência de etapas lógicas, fornece explicações explícitas sobre o raciocínio e o processo de pensamento em cada estágio (Del & Fishel, 2022). Outra técnica que busca enriquecer os *prompts* para fornecer mais contexto é o “generated knowledge” (Liu et al., 2021), que aproveita a capacidade das LLMs para gerar informações potencialmente úteis sobre uma determinada pergunta ou *prompt*

antes de gerar uma resposta final. Na mesma linha, existe a "tree of thoughts" (ToT) (Yao et al., 2024), que é uma abordagem estruturada para guiar os LLMs explorando múltiplos caminhos de raciocínio. Ao contrário dos *prompts* lineares tradicionais, o ToT permite que os LLMs considerem várias soluções e estratégias possíveis, incluindo olhar para a frente, retroceder e fazer autoavaliações, tornando-o mais interativo e adaptável à complexidade da tarefa em questão.

Existem também trabalhos que buscam catalogar os *prompts* (White et al., 2023) para derivar padrões que podem ser usados de forma sistemática no projeto das estratégias de prompts. Outra possibilidade é usar a própria LLM para criar e ajustar os *prompts* antes de usá-los, em um processo de otimização de *prompts*, ajustando sistematicamente a precisão e a relevância deles, reduzindo a necessidade de tentativa e erro manual. Os principais usos de LLMs para geração de código podem ser classificados (L. Chen et al., 2024) como: (1) a tradução de requisitos em linguagem natural para código executável, que envolve a conversão de descrições de funcionalidades ou histórias de usuário em *scripts* de programação; (2) o preenchimento de *templates* de código, completando trechos de código parcialmente escritos por meio da adição das partes faltantes baseadas no contexto; (3) a refatoração e otimização de código existente, modificando a estrutura e eficiência do código sem alterar sua funcionalidade; e (4) a geração de casos de teste, criando *scripts* e cenários para validar a correção e o desempenho do código. Neste trabalho, utilizamos as três primeiras técnicas, mas o foco maior dos experimentos foi na avaliação da primeira técnica.

Para a avaliação das ferramentas, além da interatividade, aparência e funcionalidades, usamos como métrica base a taxa de sucesso na compilação ou interpretação, que constitui uma métrica fundamental para avaliar a qualidade da geração de código, verificando se o código produzido pode ser compilado ou interpretado sem erros sintáticos (C. Chen et al., 2020).

Apesar dos benefícios potenciais, os LLMs também impõem desafios para os educadores que buscam utilizá-los na geração de ferramentas de ensino. (Chu et al., 2025b). Um dos principais obstáculos é a validação pedagógica e técnica dos materiais produzidos, uma vez que o conteúdo gerado pode conter erros conceituais, inconsistências lógicas ou soluções tecnicamente incorretas. Além disso, há a dificuldade de manter a coerência didática entre diferentes ferramentas produzidas por modelos de linguagem distintos, o que exige dos docentes um papel mais ativo na curadoria, revisão e adaptação dos recursos gerados. Outro desafio relevante diz respeito à transparência e explicabilidade dos modelos, já que compreender as decisões e limitações das LLMs é essencial para garantir que as ferramentas resultantes apoiem efetivamente o processo de ensino-aprendizagem. Por fim, questões como dependência tecnológica, custo computacional, privacidade dos dados utilizados nos *prompts* e formação docente para o uso crítico dessas tecnologias configuram barreiras práticas à adoção sustentável dos LLMs na educação. Tais desafios indicam a necessidade de um olhar mais reflexivo e estratégico por parte dos educadores, que devem equilibrar o potencial criativo e produtivo dessas ferramentas com a responsabilidade pedagógica e ética de sua aplicação no contexto educacional (L. Yan et al., 2024).

4 Metodologia

Neste trabalho, avaliamos várias técnicas para a utilização de modelos LLMs no desenvolvimento de ferramentas educacionais. O primeiro passo foi realizar uma análise exploratória da capacidade de geração de interfaces gráficas com *prompts* simples especificados por usuários diferentes para

a mesma tarefa. Por exemplo, solicitamos a criação de uma interface para o ensino de árvores binárias, onde diferentes usuários formularam requisições ligeiramente distintas para o mesmo objetivo educacional. Mostramos que, a partir de um *prompt* de 3 a 8 linhas, é possível convergir para o código de uma ferramenta didática com visualização gráfica e interativa. Avaliamos a capacidade de geração e compreensão de quatro LLMs que atenderam às requisições de cinco usuários para os mesmos problemas, seguindo caminhos ligeiramente distintos nas requisições. Para avaliar as quatro LLMs (ChatGPT, Claude, Copilot e Gemini), o experimento envolveu a geração de um conjunto de dados com mais de 350 ensaios, contemplando pelo menos nove problemas distintos, usando no mínimo quatro LLMs e no mínimo duas técnicas distintas na escrita dos *prompts*.

Selecionamos um conjunto de 9 temas comuns em fundamentos de ciência da computação baseado em trabalhos recentes sobre ferramentas de ensino (Julio et al., 2024): ordenação por inserção, ordenação por seleção, inserção/remoção (I/R) em árvore binária, I/R em heap, I/R em tabela hash, menor caminho, árvore geradora mínima, máquina de Turing e o jogo da velha. Apesar do jogo da velha ser uma exceção entre os temas, ele representa uma matriz como base para manipulação, que é um tópico presente no ensino. Além dos 9 exemplos básicos, avaliamos também alguns exemplos de outros tópicos para avaliar isoladamente outros aspectos.

Cinco usuários, sendo 1 docente e 4 discentes, elaboraram os *prompts*. Cada usuário trabalhou de forma independente, o que nos permitiu verificar a consistência das ferramentas. Apesar de a ideia básica do *prompt* ter sido compartilhada, cada usuário fez a descrição com seu próprio estilo, criando *prompts* simples de 3-8 linhas.

Além disso, cada usuário avaliou uma técnica de *prompt* avançada diferente, como *prompt* detalhado (B. Chen et al., 2023), *zero shot Chain-of-Thought* (Wei et al., 2022), *golden Chain-of-Thought* (Del & Fishel, 2022), *generated knowledge* (Liu et al., 2021) e *least-to-most* (Zhou et al., 2022). Todos os experimentos foram documentados e estão disponíveis publicamente. Devido a restrições de espaço, apresentaremos apenas alguns exemplos e um resumo dos principais resultados; maiores detalhes estão disponíveis na documentação suplementar (UFV, 2025).

É importante destacar que, ao variarmos ligeiramente as descrições com usuários diferentes, testamos a capacidade e a reprodutibilidade das ferramentas durante um período de 2 meses. Como as ferramentas estão em constante atualização e aprendem com o perfil de cada usuário, esse aspecto também influenciou o desempenho. Nosso objetivo é testar a robustez em diferentes cenários, considerando a versão sem assinatura, pois é provável que ocorra uma maior restrição de recursos sem assinatura devido aos altos custos de uso de LLMs (Khowaja et al., 2024).

Para além da estrutura experimental, a condução deste estudo foi inspirada em abordagens metodológicas já consolidadas na literatura recente sobre avaliação de LLMs, que combinam análise comparativa de desempenho e métodos empíricos de inspeção qualitativa. Trabalhos como os de (Kazemitabaar et al., 2023; Mutanga et al., 2025), por exemplo, também exploram a geração automática de código e de conteúdo educacional, adotando métricas similares às utilizadas neste estudo, como taxa de compilação, completude funcional e clareza de interface, para examinar a efetividade dos modelos em tarefas práticas de ensino. Além disso, a estrutura em etapas sucessivas de experimentação e refinamento adotada aqui alinha-se às práticas descritas por (Chu et al., 2025a; Hu et al., 2025), que propõem ciclos iterativos de avaliação para compreender como os LLMs respondem a diferentes estilos e níveis de detalhamento de *prompting*. Dessa forma, a

metodologia aqui proposta não apenas segue uma lógica exploratória, mas também se ancora em evidências e estratégias metodológicas discutidas na literatura sobre avaliação aplicada de modelos generativos no contexto educacional.

Após o estudo exploratório da geração de código para interfaces visuais de ensino, optamos por investigar a capacidade das LLMs em tópicos que podem complementar as ferramentas visuais.

O segundo passo deste trabalho foi avaliar o comportamento dos mesmos *prompts* para geração de JavaScript. Observamos que dois modelos se destacaram e focamos nossos experimentos neles: Claude e ChatGPT, sendo que o modelo Claude demonstrou desempenho superior. Nesta etapa, não realizamos experimentos exaustivos, apenas comparamos visualmente as duas abordagens, onde apenas um usuário executou os *prompts*. Embora a linguagem Python tenha maior suporte das LLMs, o JavaScript é mais interessante para determinadas interfaces por ter sido desenvolvido especificamente para navegadores. Além disso, o JavaScript não requer conexão com o Google Colab e pode executar localmente, constituindo um recurso a ser explorado. Os códigos gerados são consideravelmente maiores devido às extensas partes repetitivas com trechos HTML, o que justifica o uso de LLMs, que são adequadas para gerar esses segmentos.

O terceiro passo verificou a capacidade das LLMs na tradução de código de uma linguagem para outra (L. Chen et al., 2024). Como geramos muitos exemplos em Python, selecionamos os códigos de dois usuários para o experimento. Avaliamos a tradução do código Python para JavaScript e, novamente, o modelo Claude se destacou.

O quarto passo foi voltado para aumentar as formas de interatividade, visando mostrar os exemplos de forma gráfica e complementar as ferramentas existentes. A geração de exemplos aleatórios foi o recurso explorado na primeira parte deste trabalho. Entretanto, é interessante oferecer uma janela de edição onde o estudante ou o professor possam modificar graficamente o exemplo. Comparamos o ChatGPT e o Claude em um experimento de edição de grafos, onde o Claude apresentou um editor funcional com poucos *prompts*.

O quinto passo consistiu em investigar o uso de linguagens de domínio específico como ferramenta de ensino. Linguagens específicas simplificam a abstração ao oferecer comandos de alto nível para ensino de uma determinada área, como tópicos de estrutura de dados (por exemplo, grafos ou árvores) ou tópicos de inteligência artificial (como aprendizado de máquina). Podemos criar comandos em português, o que pode ser interessante para públicos que não dominam a programação. O usuário pode criar facilmente um grafo em uma linguagem dedicada e executar algoritmos e avaliar algumas métricas. A questão é: como podemos usar as LLMs nesta tarefa e quais as vantagens? Uma vez criada e validada uma linguagem, os exemplos gerados estarão corretos e não farão mais uso da LLM, pois o usuário irá programar na nova linguagem. Essa abordagem evita dois problemas principais. Primeiro, o custo computacional do uso das LLMs, onde o modelo é usado apenas uma vez para criar o *parser* e o gerador. Segundo, se para cada exemplo acionarmos a LLM, temos o risco de alucinações e a possibilidade de erros, pois precisamos verificar cada geração. Nessa linha, criamos uma linguagem para geração de grafos, dando continuidade ao exemplo desenvolvido em um trabalho anterior sobre aprendizado de máquina (Coura et al., 2025).

O sexto passo foi direcionado para explorar uma grande questão sobre a dependência do uso das LLMs e a falta de interesse dos estudantes em querer entender o que foi gerado (Mutanga

et al., 2025), limitando-se apenas a usar sem compreender. Apresentamos alguns exemplos de *prompts* para uso da LLM na análise e comparação de códigos, visando motivar os estudantes. A ideia é que eles busquem entender como o problema está sendo resolvido e quais recursos de programação foram utilizados, sem se preocupar inicialmente com detalhes como gramática, bibliotecas ou formato dos dados (Y.-M. Yan et al., 2025). Estudos recentes (Lyu et al., 2025) mostram que o uso da IA para trabalho em grupo com análise de códigos é mais efetivo do que o uso sem LLM ou trabalho individual com LLM.

Finalmente, o sétimo passo foi direcionado para uma tarefa na qual as LLMs já vêm sendo muito utilizadas, com o intuito de reduzir o esforço docente e padronizar a elaboração de questionários para acompanhar as atividades. Alguns *prompts* foram elaborados e validados até a geração de um *template* com formato definido de entrada para questionários de múltipla escolha. A entrada é um JSON bem definido, que pode ser facilmente utilizado pela LLM para gerar novas questões. Também utilizamos as LLMs para gerar um formulário que permite preencher as questões, caso o professor prefira este estilo para elaborar mais questões.

A partir dessas etapas, os resultados apresentados na seção seguinte ilustram de forma prática como as estratégias metodológicas adotadas permitiram avaliar o potencial dos LLMs na geração de ferramentas educacionais interativas e visuais, revelando padrões, limitações e oportunidades observadas ao longo do processo experimental.

5 Ferramentas Geradas

5.1 Configurações

Os modelos de linguagem comerciais foram utilizados no período de Outubro de 2024 à Agosto de 2025, contemplando as seguintes versões: ChatGPT 3.5 e 4.0, Claude 3.7, Copilot o3-mini, Gemini 2.5 e DeepSeek 3.1. Por se tratarem de ferramentas comerciais, não há controle sobre eventuais variações de versões ao longo do período.

Com o objetivo de avaliar diferentes contextos de uso, cinco usuários realizaram os experimentos em ordens variadas. Embora não haja controle sobre as configurações internas dos modelos, uma vez que são sistemas em constante atualização, todos os *prompts* utilizados foram armazenados e estão disponíveis no repositório associado ao artigo. Não foram empregados parâmetros adicionais nas chamadas, e tanto a quantidade quanto a sequência dos *prompts* de cada experimento podem ser consultadas no repositório (UFV, 2025).

Por se tratarem de ferramentas probabilísticas, a reprodução exata dos resultados está sujeita a variações. O uso de modelos de código aberto permitiria maior controle experimental, porém demandaria recursos computacionais elevados e, em geral, apresenta desempenho inferior aos modelos comerciais. Assim, apesar da limitação na reprodutibilidade estrita, optou-se por uma abordagem exploratória baseada no acesso gratuito a ferramentas comerciais de estado da arte no período do experimento, possibilitando uma "fotografia" do cenário, com registro completo dos *prompts* e códigos para análises futuras.

5.2 Interface Gráfica com Python

Devido à restrição de espaço, não é possível apresentar as mais de 250 ferramentas visuais geradas. Portanto, destacam-se apenas as características visuais e funcionais mais relevantes de alguns exemplos. A Figura 1 apresenta 5 interfaces para o ensino de tabela hash aberta (que permite múltiplos elementos por entrada). Nestas interfaces, o usuário pode realizar operações de inserção, remoção e consulta na tabela. As interfaces mostradas nas figuras 1(a,c,d) utilizam a biblioteca **svgwrite** para criar desenhos gráficos vetoriais e animados. Entre estas, apenas a interface da Figura 1(c) exibe a progressão quadro a quadro, enquanto as demais implementam sobreposição de elementos, criando um efeito de animação. Já as interfaces ilustradas nas figuras 1(b,e) foram desenvolvidas utilizando a biblioteca **graphviz**. Para otimizar o processo de desenvolvimento e obter resultados satisfatórios com menos iterações, a maioria dos *prompts* incluiu especificações explícitas sobre o uso destas bibliotecas gráficas.

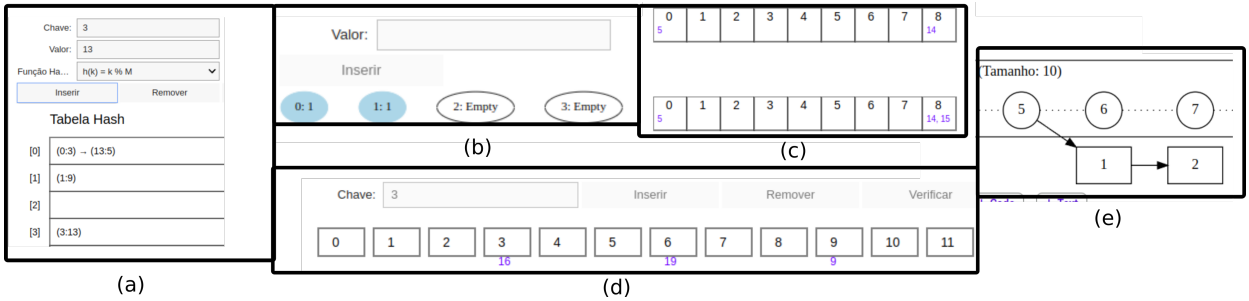

Figura 1: Hash: (a) Tabela Vertical; (b) Graphviz; (c) Horizontal quadro a quadro; (d) Horizontal com animação; (e) Graphviz com lista encadeada por entrada.

Por exemplo, um dos 10 *prompts* usados para criar uma ferramenta de visualização do *hash* está ilustrado no Quadro 1. Foram necessárias três interações (3 “shots") usando a ferramenta *chatgpt* para obter um resultado satisfatório.

Quadro 1: Exemplo de uma sequência de *prompts* para Tabela *hash*

Shot 1. Por favor, você poderia escrever um código usando iwidget para Google Colab que mostre a inserção com hash aberto em um hashset?
Shot 2. Que tal representarmos o hashset graficamente usando graphviz?
Shot 3. Agora seria legal poder remover elementos. Pode escolher se vai usar o botão de adicionar, removendo o elemento se ele já existir, ou criar um botão separado para remover elementos

A Figura 2 mostra três ferramentas adicionais geradas para o ensino de ordenação. As interfaces exibem o pseudo-código, que pode ser animado juntamente com a visualização dos elementos no vetor. O usuário pode trocar os elementos do vetor. As figuras 2(a-b) ilustram o algoritmo de inserção com a visualização do vetor, enquanto a Figura 2(c) apresenta a visualização com barras representando os valores do vetor para o algoritmo de seleção. Essas interfaces demonstram que é possível acompanhar a execução passo a passo do algoritmo, com animações tanto no pseudo-código quanto no vetor.

A Figura 3 mostra três interfaces para o jogo da velha e uma interface para a máquina de Turing. A máquina de Turing possui uma entrada dinâmica, com animação sincronizada entre a

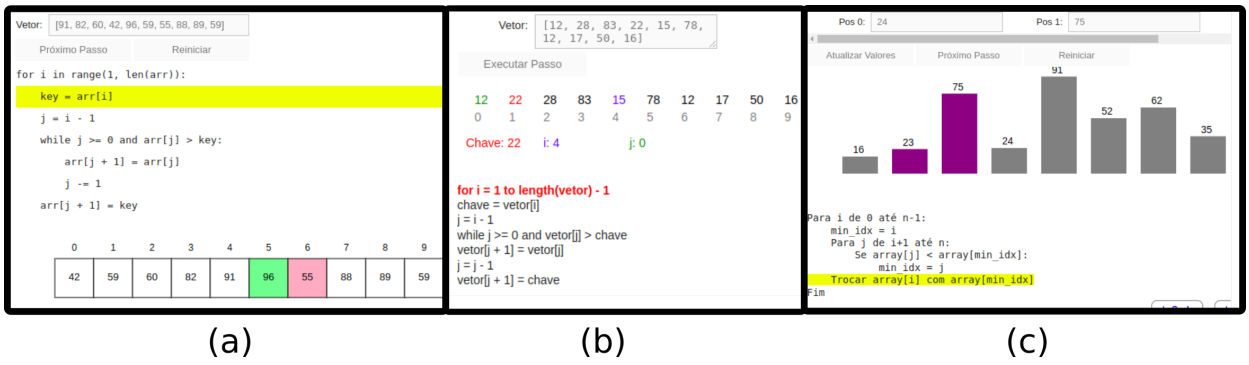

Figura 2: Ordenação com destaques: (a) Cores; (b) Indices; (c) Barras.

fita e o diagrama de estados durante o processamento de cada caractere de entrada. A máquina do exemplo está programada para incrementar em uma unidade o número binário na fita.

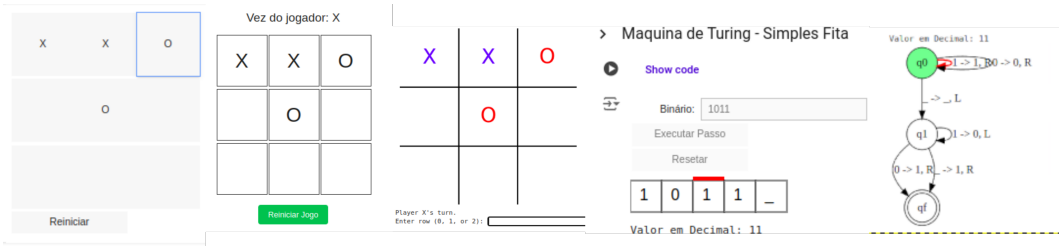

Figura 3: Três interfaces para o Jogo da Velha e uma interface para Máquina de Turing.

Para testar a capacidade para geração de jogos, solicitamos a criação do jogo resta-um ilustrado na Figura 4. Podemos observar a diversidade de soluções apresentadas com uma boa apresentação e funcionalidade, gerando mais 4 exemplos de ferramentas visuais de ensino. Como o código gerado é comentado e estruturado, pode ser usado como material didático para gerar extensões ou adaptações para outros jogos similares.

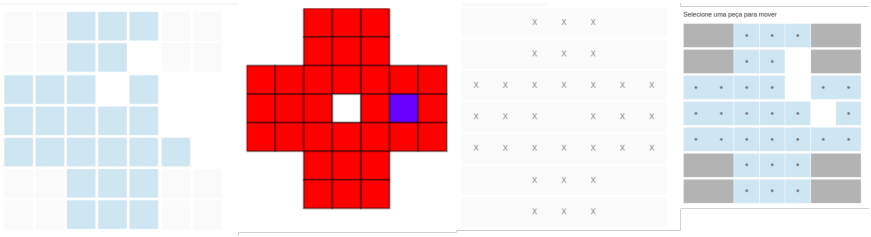

Figura 4: Quatro interfaces para o jogo Resta-um.

A Figura 5 mostra duas interfaces para árvore geradora, a primeira mais simples com destaque apenas para as arestas, a segunda mostra os agrupamentos sendo formados pelo algoritmo de Kruskal e a visualização do pseudo-código.

A Figura 6 mostra três ferramentas adicionais que destacam a versatilidade das LLMs na geração de interfaces e ferramentas de simulação. A Figura 6(a) exibe uma máquina de estados construída a partir da tabela de transição, juntamente com o simulador. A Figura 6(b) ilustra o

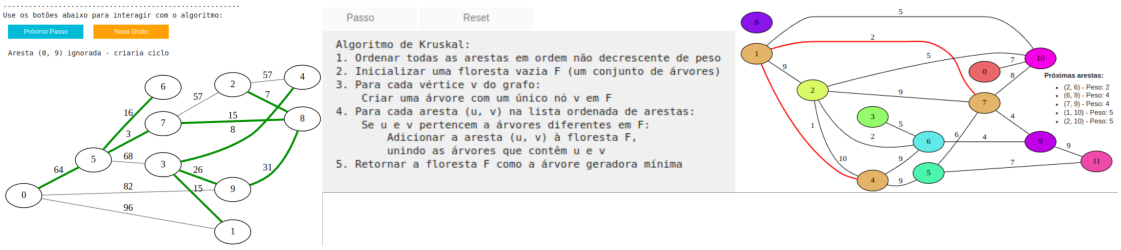

Figura 5: Duas interfaces para exemplos de Árvore Geradora Mínima.

ensino de inserção e remoção em uma lista encadeada, e a Figura 6(c) apresenta um simulador de RISC-V gerado a partir de um *prompt* com menos de 100 palavras, incluindo um campo para editar o código e simular um subconjunto de 15 instruções básicas que não foram descritas no *prompt*.

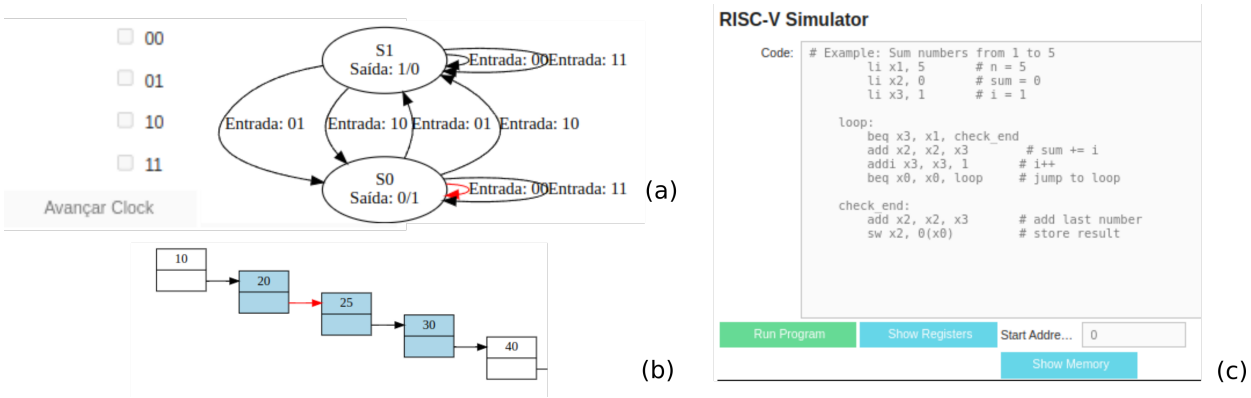

Figura 6: (a) Máquina de Estados; (b) Lista Encadeada; (c) Simulador Risc-V.

Outro resultado é gerar material para extensões. A Figura 7 mostra extensões para ensino de árvore binária onde a ferramenta ilustra o antes e depois da inserção e remoção, além do caminho para uma busca e como a árvore pode ser armazenada em um vetor.

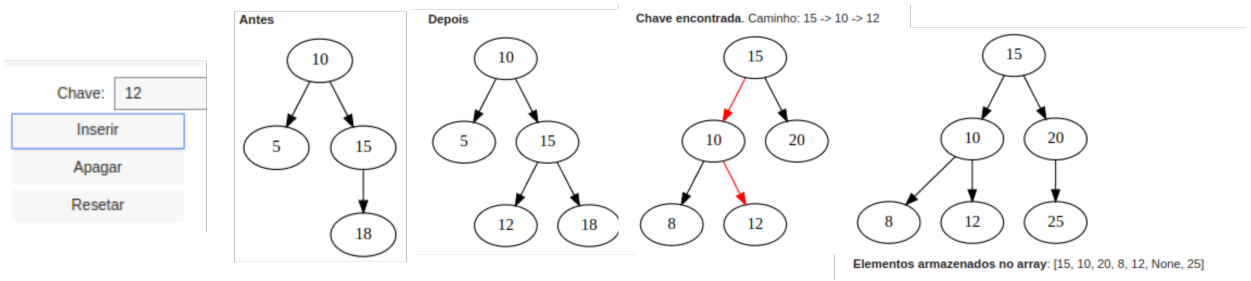

Figura 7: Extensão da Ferramenta Árvore Binária.

5.3 Geração com Javascript

Avaliamos os quatro modelos para o exemplo da inserção e remoção em árvore binária. O ChatGPT gerou uma interface simples e funcional com 140 linhas de código a partir do primeiro

prompt. O modelo Claude gerou a melhor resposta: a partir do mesmo *prompt*, produziu um código com 792 linhas (Figura 8). Vale ressaltar que, em JavaScript, o código é maior devido aos trechos em HTML inseridos, que possuem muita redundância. Este código gerado possui várias funcionalidades adicionais, com uma barra lateral e descrição textual do processo de inserção na árvore. O Copilot precisou de três tentativas para gerar um código com alguma funcionalidade, resultando em um código compactado para JavaScript com apenas 198 linhas. O Gemini gerou um código funcional, porém sem exibir as arestas que conectam os vértices das árvores; o segundo *prompt* gerou um código com 678 linhas.

Quadro 2: *prompt* para Árvore Binária em Javascript

Shot 1. Faça o código em javascript de um simulador para operações de inserção e remoção em uma estrutura de dados árvore binária com interface interativa com botões e visualização passo a passo.

Para esta tarefa, o modelo Claude de acesso gratuito se mostrou bem superior na geração de JavaScript, mesmo para um *prompt* simples ilustrado no Quadro 2. O ChatGPT gerou um código funcional e compacto, mas com uma interface bem menos atrativa que a do Claude. O Gemini gerou uma interface atrativa, mas inferior à do Claude. Dando continuidade ao experimento, avaliamos a geração de exercícios com interface gráfica nos dois modelos que foram mais bem-sucedidos: Claude e Gemini.

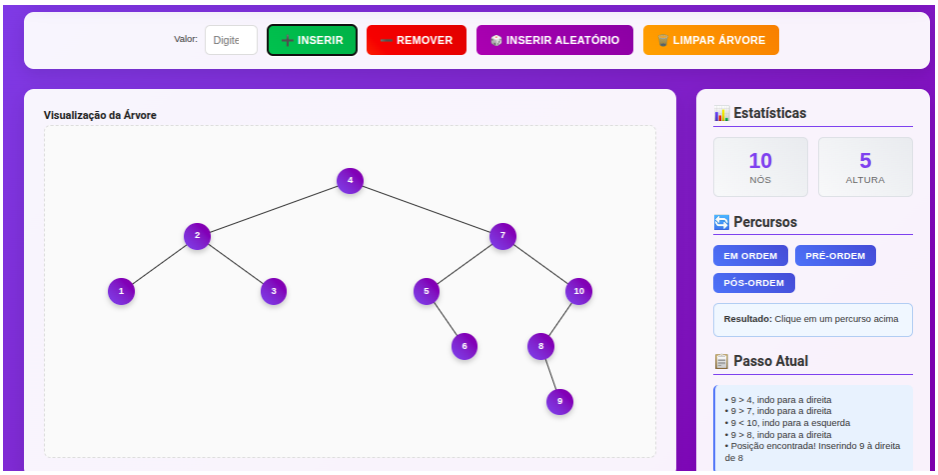

Figura 8: Interface para Árvore Binária com JavaScript. Barra lateral com informações sobre números de nós e altura, opções para percursos de travessia da árvore e aba com explicações do processo de inserção do item na árvore.

No caso dos exercícios com interface gráfica, o Claude obteve um desempenho bem superior, como ilustrado na Figura 9(a), onde foi necessário o uso de dois *prompts* curtos com 37 e 53 palavras, respectivamente, resultando em um código com 741 linhas. O Gemini não compreendeu a geração gráfica das opções de resposta e gerou uma saída em texto. Realizamos então um segundo teste com o Claude para avaliar sua capacidade de correção do questionário do Gemini e geração de uma interface gráfica com a resposta. O Claude foi bem-sucedido, como ilustra a Figura 9(b): além da visualização ilustrada, forneceu explicações detalhadas. Elaboramos exemplos semelhantes com exercícios de heap.

Como o modelo Claude se mostrou significativamente superior aos demais, investigamos seu desempenho para os problemas estudados em Python, além da árvore binária e heap. Foram

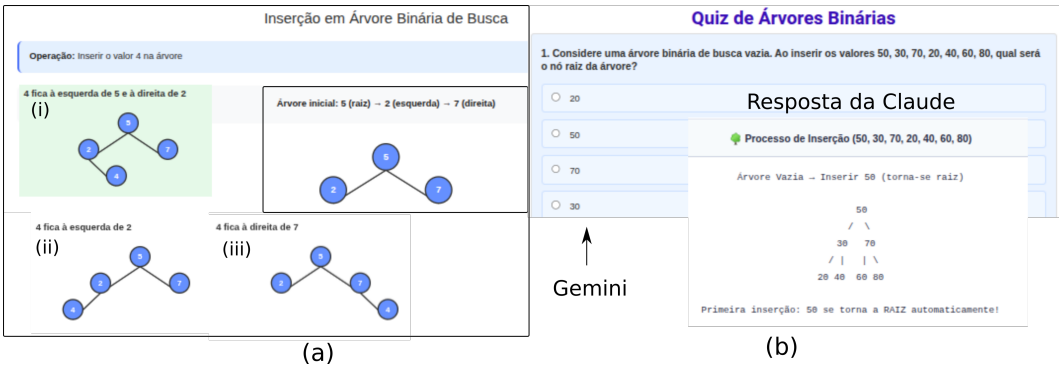

Figura 9: Interface para exercícios com visualização e correção para Árvores Binárias com Javascript: (a) Exemplo gerado pelo modelo Claude com três opções visuais de resposta para inserção do item 4 na árvore em destaque; (b) Exemplo Textual da Gemini com correção visual elaborada pelo Claude.

avaliados os seguintes problemas: *hash*, ordenação por seleção e inserção, máquina de Turing, algoritmos de Dijkstra e Kruskal. Mais da metade dos exemplos foram gerados com apenas um *prompt*; a média foi de 1,4 *prompts* e o pior caso, de 3 *prompts*.

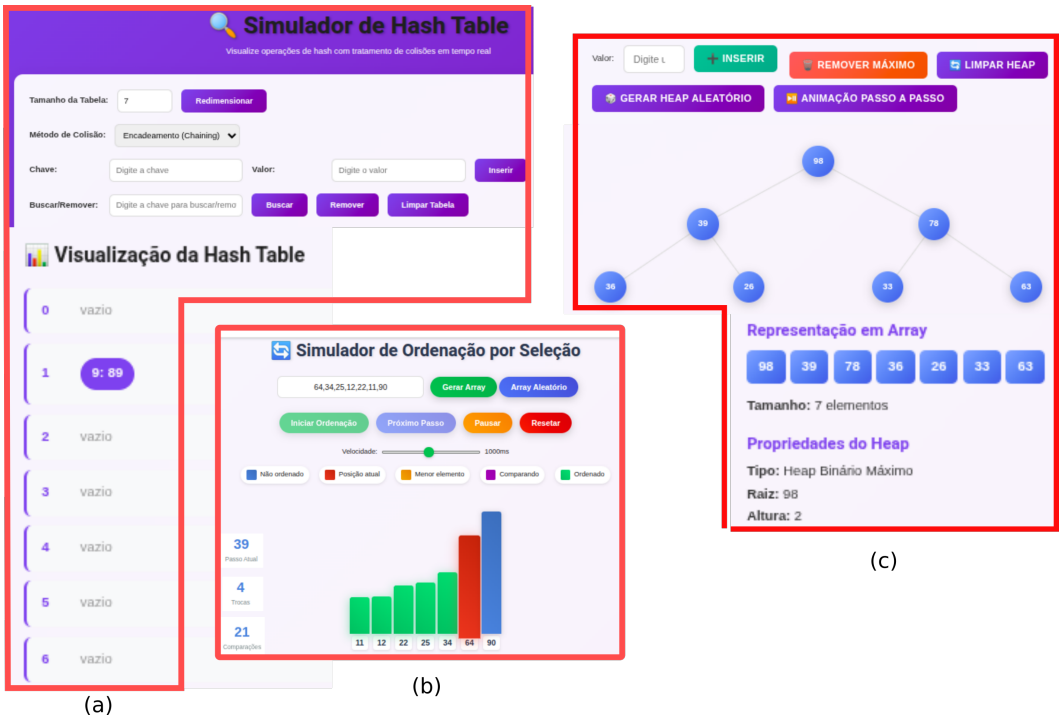

Figura 10: Interface com visualização gerada pelo modelo Claude em Javascript: (a) Inserção e Remoção em *hash* ; (b) Algoritmo de Ordenação por Seleção; (c) Heap.

A Figura 10 ilustra três dos exemplos: *hash* , ordenação por seleção e heap. Nota-se um estilo de cores mais vivas e o uso das cores para destaques com legendas automáticas. No caso do exemplo de heap, além da visualização em forma de árvore, foi gerado um botão para exemplos aleatórios que já inicializam o heap com tamanho delimitado, além de mostrar a implementação em um vetor e algumas estatísticas. Nos outros dois exemplos da tabela *hash* e do algoritmo de seleção também são ilustradas algumas estatísticas que não estão na figura devido a restrições de espaço.

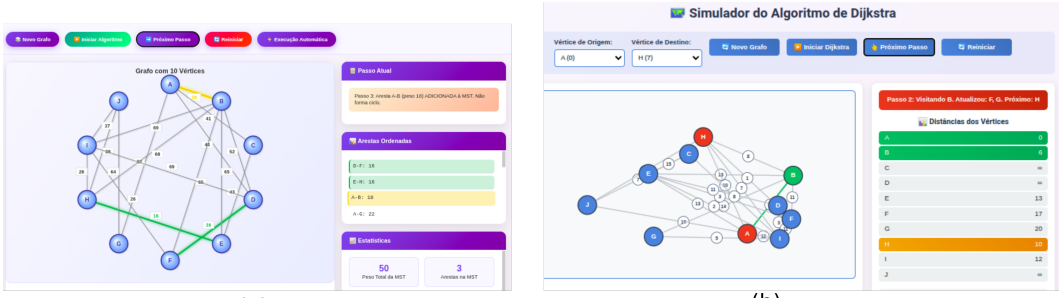

Figura 11: Interface com visualização gerada pelo modelo Claude em Javascript: (a) Algoritmo de Kruskal; (b) Algoritmo de Disjktra.

A Figura 11 ilustra os exemplos de algoritmos com grafos. Neste caso, a interface gerada tem uma janela de visualização à esquerda e a aba da direita ilustra detalhes importantes para a compreensão da execução do algoritmo. Para o algoritmo de árvore geradora mínima de Kruskal, a aba da direita mostra quais arestas já foram visitadas e seus pesos. Para o algoritmo de Dijkstra, o vetor de distâncias, que é atualizado ao longo da execução, é mostrado a cada passo. Além disso, ambos possuem a parte de navegação com a execução passo a passo e a geração de exemplos na parte superior. Esta interface atrativa em JavaScript nos motivou a avaliar dois outros recursos didáticos: uma janela de edição para manipulação ainda mais interativa dos exemplos e linguagens de domínio específico para elaboração de exemplos e casos de teste clássicos dos algoritmos de grafos. Ambos os exemplos estão ilustrados nas próximas seções.

Um ponto importante a ser ressaltado é que todos os códigos JavaScript gerados e incluídos no Google Colab executaram sem a necessidade de conexão aos servidores do Google, ou seja, executam localmente, o que é uma vantagem em relação ao Python, pois em ambientes com muitas conexões simultâneas em laboratórios presenciais podem ocorrer atrasos devido à sobrecarga nos serviços do Google. Este problema pode ser contornado com um servidor local, caso ocorra, pois os *notebooks* podem ser executados no Colab ou em ambientes locais.

5.4 Tradução Python para Javascript

Outro teste realizado foi avaliar a capacidade de tradução de linguagens fonte para fonte. Como o JavaScript se mostrou atrativo, investigamos o desempenho das LLMs na tradução dos códigos Python de criação de interface gráfica, gerados em etapas anteriores do estudo, para a linguagem JavaScript. O objetivo principal foi analisar se os modelos conseguiam converter ferramentas educacionais interativas, originalmente concebidas para o ambiente Google Colab utilizando Python, em aplicações equivalentes e funcionais no ambiente do navegador, aproveitando as capacidades das células de código `%%html` e `%%javascript` do Colab.

Para realizar este experimento, selecionamos os códigos Python desenvolvidos para a visualização da estrutura de dados de árvore binária. Esses códigos-fonte, resultantes das iterações com as quatro LLMs de dois alunos diferentes (aluno 1 e aluno 3) na fase de geração em Python, foram utilizados como entrada para o processo de tradução.

Um conjunto de LLMs foi empregado para a tarefa de tradução, além das quatro originais. Para esse experimento, foram acrescentados Deepseek e Gemini PRO. A cada modelo, forneceu-

se um código Python e um *prompt* direto para realizar a tradução para JavaScript, sem fornecer exemplos de como a tradução deveria ser feita (abordagem *zero-shot*).

Durante a avaliação das respostas das LLMs, adotou-se a seguinte regra para refinar as tentativas: caso a primeira resposta do modelo à solicitação de tradução não resultasse em nenhum elemento visual na interface gerada, uma segunda tentativa era realizada com o mesmo *prompt*. Se, mesmo após esta segunda tentativa, nenhum componente visual fosse apresentado, considerava-se que a tradução para aquele código específico não havia sido bem-sucedida.

Para ilustrar o processo de tradução e a diversidade de saídas obtidas, incluímos dois exemplos específicos. A Figura 12 apresenta a saída do código gerado pelo modelo DeepSeek ao traduzir um código Python originalmente produzido pelo ChatGPT, a qual manteve um design mais básico, porém totalmente funcional. A Figura 13 exibe o resultado da tradução realizada pelo próprio modelo Claude sobre um código Python que ele mesmo havia gerado anteriormente, que se destaca pelo uso de cores, sendo visualmente mais elaborado. É possível verificar que diferentes casos de sucesso podem gerar saídas gráficas significativamente distintas.

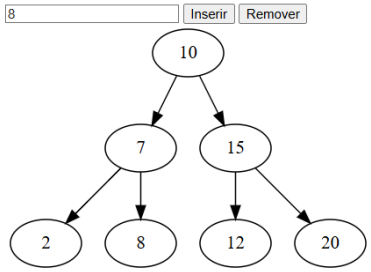

Figura 12: Tradução de Python (GPT) para JavaScript usando DeepSeek.

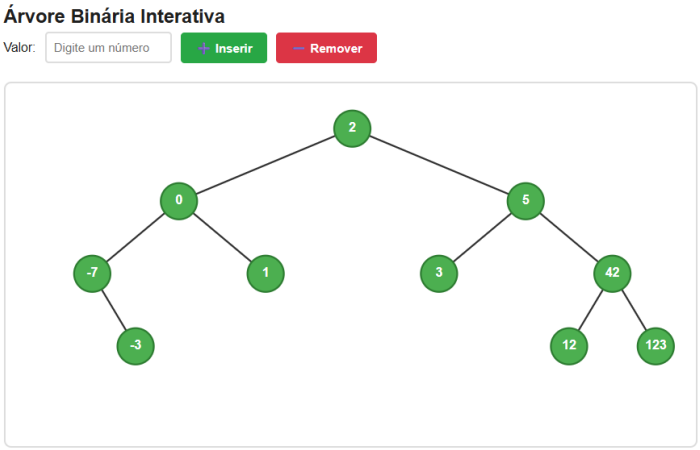

Figura 13: Tradução de Python (Claude) para JavaScript usando o próprio Claude.

Esta metodologia permitiu avaliar a capacidade de diferentes LLMs não apenas em realizar a conversão sintática entre as linguagens, mas também em preservar a funcionalidade e a interatividade das ferramentas educacionais no novo ambiente de execução.

5.5 Recursos de Edição

A maioria dos exemplos anteriores com grafos e árvores foi baseada em uma interface que, passo a passo, de forma textual, solicitava a inclusão de novos vértices ou, através da interface gráfica, gerava um grafo aleatório inicial. Como existem muitas ferramentas de interface com recursos de edição, fizemos o experimento para avaliar se a LLM Claude é capaz de gerar um editor gráfico. Inicialmente, testamos a capacidade em Python e depois na combinação Python e JavaScript. Porém, em ambos os casos, não foi possível a geração de um código funcional, limitando-nos a no máximo 5 prompts. Solicitamos então o uso de JavaScript, que se mostrou visualmente mais atrativo.

Quadro 3: *prompts* para o Editor de Grafos em Javascript

Prompt 1: Faça um código em javascript que será executado no ambiente google colab com interface gráfica que permita ao usuário criar um grafo, adicionar arestas e depois escolher um algoritmo para executar, de forma visual, como árvore geradora e busca em largura ou profundidade.
Prompt 2: ok, mas gostaria que executasse apenas os algoritmos de dfs, bfs e árvore geradora mínima (kruskal); além disso, gostaria que os pesos não fossem aleatórios, mas definidos pelo usuário. Também há um problema em que o espaço para montagem do grafo está limitado apenas à metade do que está visível; gostaria que fosse possível utilizar todo o espaço do "quadro" para montar o grafo.
Prompt 3: gostaria que pudesse também inserir arestas de um vértice para ele mesmo.
Prompt 4: agora poderia acrescentar uma melhoria para que fosse possível ao usuário mover os vértices de posição dinamicamente?

Com quatro prompts, totalizando 136 palavras (veja o Quadro 3), foi possível gerar uma ferramenta funcional com 816 linhas de código. O editor possui recursos para inserir vértices e arestas usando o *mouse*, assim como para remover e mover. O movimento permite o arraste das conexões. A interface possui três algoritmos de grafos que podem ser visualizados passo a passo; o estudante pode acrescentar ou remover vértices e executar os algoritmos novamente.

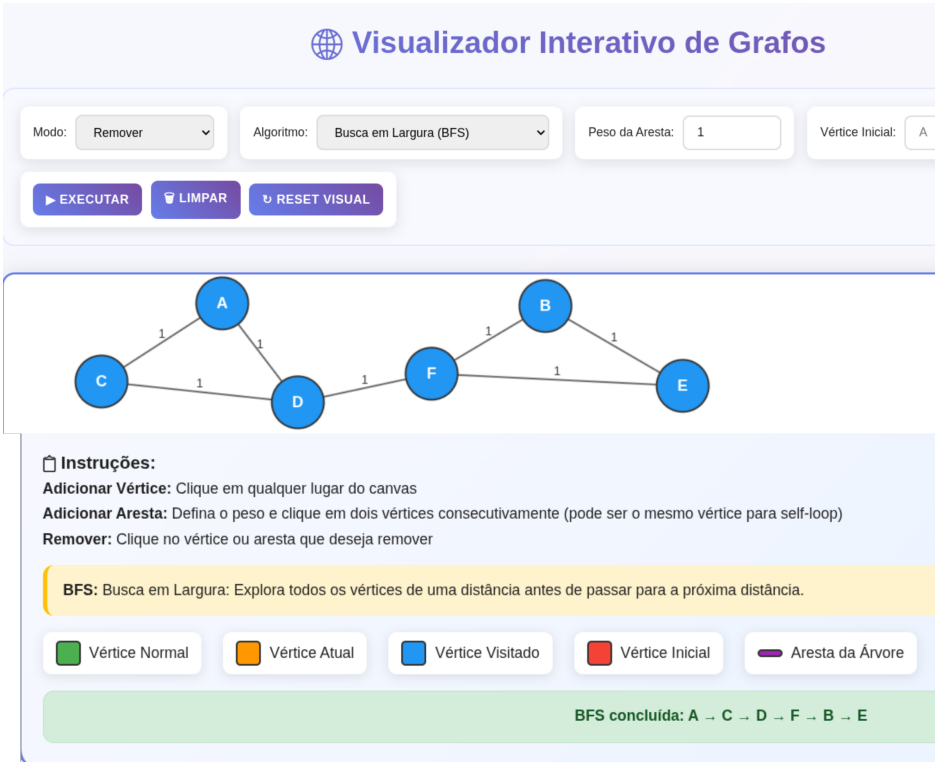

Figura 14: Editor de Grafos com interface interativa gerada pelo modelo Claude em Javascript.

A Figura 14 ilustra um exemplo de uso do editor, onde foi desenhado um grafo com 6 vértices e 7 arestas para depois executar o algoritmo de busca em largura a partir do vértice A.

5.6 Linguagens de Domínio Específico

Em virtude de linguagens de programação serem intrinsecamente complexas – devido à variabilidade, generalidade e completude de suas construções – docentes de instituições de ensino superior frequentemente enfrentam dificuldades (Barbosa et al., 2016) ao ensinar tópicos especializados, exigindo o conhecimento de bibliotecas e detalhes de implementação em uma determinada linguagem. Os tópicos podem variar de teoria dos grafos a análises avançadas de dados. Tradicionalmente, essas disciplinas requerem que os estudantes dominem simultaneamente conceitos teóricos complexos e sintaxes específicas de linguagens como Python, R ou Java. Uma alternativa é o uso de linguagens específicas semelhantes à pseudo-linguagem Portugol, que é amplamente usada em livros em português (Zanini & Raabe, 2012).

Apesar de romper a barreira do inglês, o Portugol, sendo produto de linguagens como ALGOL e Pascal, oferece uma transição limitada, pois sua sintaxe genérica não captura a essência de domínios específicos como grafos ou aprendizado de máquina. Esta limitação motiva o desenvolvimento de Linguagens de Domínio Específico (DSLs) educacionais que possam abordar conceitos avançados com sintaxe intuitiva e contextualmente apropriada para cada área de conhecimento.

Nossa proposta consiste em avaliar o desenvolvimento de DSLs em português para diferentes áreas da computação, utilizando LLMs para gerar automaticamente os interpretadores dessas linguagens. Para teoria dos grafos, comandos como *criarGrafo()* ou *adicionarVertice("A")* permitem que estudantes se concentrem nos algoritmos fundamentais. Para aprendizado de máquina, instruções como *carregarDados("dataset.csv")* ou *treinarModelo("regressaoLinear")* abstraem complexidades sintáticas irrelevantes ao domínio.

Os LLMs eliminam a necessidade de conhecimento técnico profundo em *parsing* e geração de código, tornando a abordagem acessível para educadores de diversas especialidades no desenvolvimento de linguagens didáticas para tópicos específicos. Esta democratização permite que professores criem ferramentas pedagógicas personalizadas sem dominar técnicas complexas de desenvolvimento de compiladores.

Em trabalho anterior (Coura et al., 2025), demonstramos esta metodologia com uma DSL para aprendizado de máquina onde o fluxo *carregarDados("iris"); selecionarColuna("especie"); escolherModelo("knn"); treinarModelo(); avaliarDesempenho()* representa a sequência de etapas dos modelos de aprendizado supervisionado de forma intuitiva. O processo inicia com a leitura de dados, seguido pela definição da variável de predição, seleção do algoritmo, treinamento e avaliação. Cada comando encapsula uma etapa conceitual fundamental, permitindo que estudantes compreendam o fluxo do aprendizado de máquina sem se perderem em detalhes de implementação de bibliotecas como scikit-learn ou TensorFlow. Esta metodologia pode ser utilizada para popularizar o aprendizado de máquina, rompendo a barreira das dificuldades com programação e complementando esforços de trabalhos recentes para inclusão de IA nas escolas (R. M. Martins et al., 2024), no ensino de ciências de forma geral (Park et al., 2023) e em todas as outras áreas do conhecimento.

Para estender a ideia com um exemplo de grafos, avaliamos as LLMs ChatGPT e Claude. A ChatGPT não foi bem-sucedido, usando 7 prompts, além de a geração visual não ser muito atrativa, pois optou por usar apenas Python e Matplotlib. O modelo Claude foi avaliado usando

uma sequência de três prompts, totalizando 260 palavras. A ferramenta gerada tem 1077 linhas de código, como ilustrado na Figura 15.

Figura 15: Linguagem de domínio específico para ensino de Grafos gerada pelo Modelo Claude em Javascript.

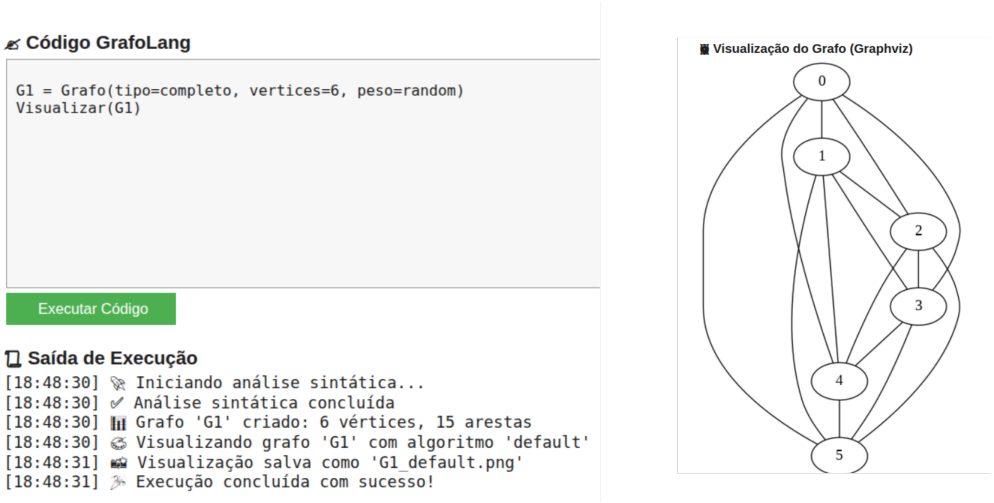

Figura 16: Linguagem de domínio específico para ensino de Grafos gerada pelo modelo Claude em Javascript.

Entretanto, diferentemente das ferramentas geradas em (Coura et al., 2025) para aprendizado de máquina em Python, a versão JavaScript não implementou um *parser* genérico; ao contrário, o tratamento da linguagem é bem específico a nível léxico da entrada. Para garantir mais portabilidade ao código gerado e futuras extensões, nossa abordagem foi solicitar a inclusão de um *parser* para ter flexibilidade a nível léxico e sintático, separando bem o *parser* e o interpretador de comandos.

Foi solicitado ao modelo Claude que executasse a transformação do código da linguagem JavaScript para a linguagem Python, que possibilita o uso da biblioteca *lark* para criar uma linguagem com gramática definida e explícita no código. Este biblioteca cria uma gramática livre de contexto com regras de derivação, símbolos terminais e não terminais, que facilita a extensão da linguagem. Apesar de o código gerado ser funcional, a primeira versão não tinha mais janela de edição nem visualização gráfica, apenas mensagens de texto. Com a experiência adquirida do estado da arte atual, em que o Claude se mostrou melhor para JavaScript enquanto o ChatGPT

para Python, optamos por fazer um novo teste. O código gerado pelo modelo Claude foi repassado para o ChatGPT, onde foi solicitada a adição de um editor e uma janela de visualização. A Figura 16 ilustra a janela de edição em Python e a visualização com Graphviz.

Apesar dos modelos LLM converterem os *prompts* em linguagem natural, para este exemplo, cada sequência de *prompts* pode gerar um interpretador diferente e pode ter limitações na análise léxica da linguagem de domínio específico que queremos gerar. Usando a metodologia de utilizar as LLMs apenas para criar o *parser* e o interpretador, podemos garantir a geração de uma ferramenta mais determinística. O professor ou aluno irá elaborar novos exemplos escritos na nova linguagem sem a necessidade de recorrer novamente à LLM, reduzindo o custo computacional e garantindo a reprodutibilidade dos exemplos.

O Quadro 4 ilustra a gramática que foi automaticamente gerada pela LLM, onde temos bem definida a linguagem para possíveis extensões. A linguagem gerada inicialmente tem seis comandos. O comando de importação e exportação pode trabalhar com vários formatos (*dot, gml, json, adjacency*). Além disso, vários algoritmos já estão incorporados como *bfs, dfs, dijkstra, kruskal, prim, bellman_ford, floyd_warshall, pagerank, clustering*, bem como diversas propriedades: grau, componentes, diâmetro, densidade, centralidade, *clustering_coef*. Isso mostra que uma combinação de modelos e poucos *prompts*, no estado atual da arte das LLMs, permite criar uma interface simples com uma linguagem de domínio específico para acesso à biblioteca *networkX* do Python para ensino de grafos e até mesmo de conceitos intermediários e avançados de redes complexas.

Quadro 4: Gramática para a linguagem DSL para Ensino de Grafos

```
GRAFOLANG_GRAMMAR = r"""
    ?start: statement+
    ?statement: assignment | visualization | export_cmd | import_cmd
    | algorithm_cmd | property_cmd
    assignment: IDENTIFIER "=" graph_creation
    graph_creation: "Grafo" "(" parameter_list? ")"
    visualization:"Visualizar" "("IDENTIFIER("," parameter_list)? ")"
    export_cmd:"Exportar" "("IDENTIFIER","export_format("," STRING)?")"
    import_cmd:"Importar" "("STRING","import_format")" "as"IDENTIFIER
    algorithm_cmd: "Executar" "(" algorithm_name ","
            IDENTIFIER ("," parameter_list)? ")"
    property_cmd: "Propriedade" "(" IDENTIFIER "," property_name ")"
    export_format: "dot" | "gml" | "json" | "adjacency"
    algorithm_name: "bfs" | "dijkstra" | "kruskal" | "prim"  | "pagerank"
    property_name: "grau" | "componentes" | "diametro" | "densidade"
    parameter_list: parameter ("," parameter)*
    parameter: IDENTIFIER "=" value
    ?value: STRING | NUMBER | IDENTIFIER | boolean
    boolean: "true" | "false" | "True" | "False"
    IDENTIFIER: /[a-zA-Z_][a-zA-Z0-9_]*/
    STRING: /"[^"]*"/ | /'[^']*'/
    NUMBER: /\d+(\.\d+)?/
    COMMENT: "//" /[^\n]*/
```

5.7 Avaliação de Código Python

Além da geração de código, as LLMs podem ser usadas para auxílio no ensino (L. Chen et al., 2024; Kazemitabaar et al., 2023) de programação para análise crítica de código. Neste trabalho, fizemos um ensaio para uso de um modelo LLM; no caso, usamos o mais popular, ChatGPT, para avaliar os códigos da própria LLM e dos outros modelos para o exemplo da ferramenta de inserção e remoção em árvore binária. Escolhemos este exemplo porque todos os modelos foram bem-sucedidos.

Tabela 1: Avaliação dos códigos gerados para árvore binária usando a ChatGPT, destaque para as diferenças entre os códigos: Gemini, ChatGPT, Claude e Copilot (Tarefa 2).

Critério	**Gemini**	**ChatGPT**	**Claude**	**Copilot**
Nomes de variáveis	Inglês	Inglês	Português	Inglês
Visualização	Básica, funcional	Clara e objetiva	Muito legível (vertical)	Clara, com formato controlado
Layout da UI	entrada juntos	Separados, com labels	Com ícones e estilização	Simples, direto ao ponto
Remoção	Lógica correta	Clara e recursiva	Ótima legibilidade	Boa lógica com rastreamento
Rastreamento de passos	x	x	x	(lista steps[])
Validação de entrada	Usa IntText (bom)	Usa Text, requer melhoria	Usa Text, conversão manual	Usa BoundedIntText
Estilo geral	Didático	Modular e limpo	Visualmente amigável	Minimalista, técnico
Reutilização de visualização	Nova a cada chamada	Nova a cada chamada	Usa atributo da classe	Recria sempre, mas bem feito

Solicitamos seis tarefas: (1) criar uma tabela com as semelhanças dos códigos; (2) identificar as diferenças; (3) destacar os melhores recursos explorados pelas LLMs; (4) selecionar e justificar a escolha do melhor código; (5) avaliar a robustez e falhas; (6) gerar a combinação dos códigos.

Para a primeira tarefa foram destacadas quatro características semelhantes: inserção e remoção na árvore, uso do graphviz, interface com widgets e atualização dinâmica da visualização. A segunda tarefa foi destacar as diferenças que estão detalhadas na Tabela 1. A tarefa 3 foi apresentar o destaque de cada LLM, onde, por exemplo, a ChatGPT teve como destaque a separação entre o código e a interface, já a Claude, a maior usabilidade e clareza visual. A quarta tarefa apresenta o melhor código e sua justificativa para escolha. O código selecionado foi da LLM Claude, tendo como critérios os aspectos educacionais, pois usou português e foi bem estruturada visualmente, já o código gerado pelo Copilot foi elogiado em termos de expansões futuras e por último o código da ChatGPT pela estrutura modular. Como mencionamos, a ChatGPT se autoavaliou e avaliou as outras LLMs, mas escolheu a Claude como melhor código em termos didáticos e não destacou a Gemini, que teve o pior desempenho em várias tarefas. A tarefa 5 avaliou a robustez, onde a ChatGPT destacou os códigos da Claude e Copilot pelo uso de try/except e BoundIntText, recursos que facilitam a depuração e validação das entradas. Por fim, a tarefa 6 fez um novo código combinando os quatro códigos, que ficou bem modular e fez uso do BoundIntText para validar a entrada numérica.

Não está no escopo deste trabalho uma avaliação detalhada do tópico, mas podemos observar que é um recurso complementar à geração da ferramenta que pode ser usado para estimular os estudantes não só a aprender o tópico em questão, como também compreender quais recursos foram utilizados pelas LLMs para geração do código de visualização. Os resultados das seis tarefas com detalhes estão disponíveis na seção 8 do Colab com o material complementar (UFV, 2025).

Essa etapa de avaliação mostrou que o uso de LLMs como agentes avaliadores de código pode contribuir não apenas para o refinamento automático das soluções geradas, mas também para o desenvolvimento do pensamento crítico dos estudantes, ao permitir a comparação de soluções distintas em um mesmo contexto educacional (L. Chen et al., 2024; Kazemitabaar et al., 2023).

5.8 Elaboração de Questionários

Existem muitas bibliotecas e ferramentas para a elaboração de questionários e, mais recentemente, vários estudos com LLMs (Kumar et al., 2025). Neste trabalho, iremos focar na geração de um *template* de pergunta para um código já estruturado, que permite a visualização no Google Colab com uma saída visual compatível com dispositivos móveis. A ferramenta foi gerada com os questionários descritos em um formato JSON, conforme ilustrado no Código 1.

```
{  "pergunta": "O que e o grau de um vertice em um grafo nao direcionado?",
    "opcoes": [
        "A quantidade de ciclos que passam pelo vertice",
        "O numero de vertices adjacentes a ele",
        "A distancia ate o vertice mais distante",
        "O numero de arestas incidentes em todo o grafo"  ],
    "correta": 1,
    "explicacao": "O grau de um vertice e o numero de arestas  que incidem
       sobre ele, ou seja, o numero de conexoes com outros vertices."
}
```

Código 1: Exemplo de estrutura JSON para questão de múltipla escolha.

Semelhante às ferramentas de linguagens de domínio específico, uma vez gerada a ferramenta, o uso da LLM fica restrito a auxiliar o professor ou aluno a gerar questões para um determinado tema para avaliação ou autoavaliação. Onde a ferramenta básica contabiliza os acertos, mostra uma explicação no caso de erro, permite um banco de questões para escolher aleatoriamente um subconjunto e tem um editor onde o professor pode gerar novas questões preenchendo os campos com uma interface simples.

Na Figura 17, visualizamos a questão descrita em JSON no Código 1. A melhor opção da interface é baseada em *ToggleButtons*, que automaticamente reformata para o tamanho de tela.

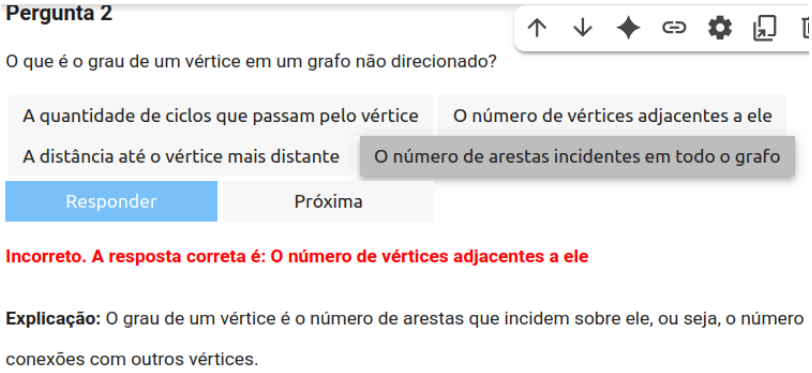

Figura 17: Exemplo de uma questão no formato JSON para o gerador de interface para questionário com vizualização em Celular.

6 Resultados Experimentais

6.1 Quantitativos para Prompts com Python

Mostraremos primeiro os resultados quantitativos, nos quais buscamos estabelecer métricas para comparar as LLMs utilizadas. A Tabela 2 apresenta um resumo com um total de 361 *prompts* para a geração de ferramentas utilizando as 4 LLMs comerciais. Obtivemos com sucesso 274 ferramentas, das quais ilustramos apenas 21 na Seção 5, devido às restrições de espaço. As colunas "erros de compilação" (**Comp.**) e "erros de execução" (**Exec.**) mostram a porcentagem de erros por tentativa de *prompt* para cada LLM. Podemos observar que as LLMs Copilot e Claude apresentaram apenas 5% e 12% de erros de compilação, e 20% e 26% de erros de execução, respectivamente. No geral, as LLMs Claude, Copilot e ChatGPT tiveram um desempenho semelhante, resultando em mais de 81,8% de ferramentas gráficas e interativas funcionais, em contraste com o Gemini, que resolveu apenas 40% dos prompts. Em termos de tamanho total dos *prompts* em palavras, após a troca de mensagens para ajustes, o Copilot foi o mais bem-sucedido, com uma média de 145 palavras por problema. Em média, 50 palavras adicionais são suficientes para ajustar o *prompt* inicial na interação com as LLMs. Devido a limitações de entrada do Gemini, poucas técnicas com *prompts* mais detalhados foram avaliadas, o que resultou em um tamanho menor dos prompts, já que os maiores não foram testados. A última coluna mostra o número de ferramentas para cada LLM que foram testadas. Por exemplo, 97 testes de geração foram realizados com a LLM Claude, que respondeu com sucesso e qualidade em 85,7% dos casos, gerando 83 ferramentas.

Tabela 2: Análise de Desempenho por Ferramenta.

	Erros		**Funcionou**	**Tam. Prompt**		
LLM	**Comp.**	**Exec.**	**Correto (%)**	**Primeiro**	**Total**	**Testes**
Claude 3.7	0,12	0,26	85,6	114,4	165,2	97
Copilot o3-mini	0,05	0,20	83.2	114,4	145,5	101
Chat Gpt 3.5	0,05	0,29	81,8	114,7	164,2	99
Gemini 2.5	0,19	0,43	40,6	81,2	154,0	64

Os resultados da Tabela 2, com acertos em torno de 80%, são compatíveis com os dados da literatura. Em um estudo de Joshi (2025), que comparou as LLMs Qwen3-Coder, Claude e DeepseekR1 para tarefas de codificação e raciocínio, observou-se uma taxa de sucesso entre

60% e 80% usando quatro conjunto de testes: SWE-bench, HumanEval, MBPP e GSM8K. De forma similar, Phogat et al. (2025)compararam as LLMs ChatGPT, Claude, Deepseek e Qwen e reportaram uma taxa de sucesso em torno de 80% nos conjuntos de teste MMLU e HumanEval.

A Tabela 3 apresenta os resultados por problema, excluindo-se a LLM Gemini devido ao seu desempenho insatisfatório. Cada problema foi submetido a avaliação com no mínimo 29 *prompts* distintos, com alguns recebendo testes adicionais, mantendo um padrão de comportamento consistente entre as avaliações. A máquina de Turing revelou-se o desafio mais complexo, apresentando uma taxa de sucesso de 56.7%, enquanto a implementação da árvore binária alcançou o melhor desempenho, com 100% de sucesso. O desempenho médio geral situou-se em 84%. O número máximo de iterações em um *prompt* foi limitado a um intervalo entre cinco e oito tentativas. Consideramos o processo sem sucesso quando há ausência de resultados satisfatórios, e então interrompemos o processo. O problema do caminho mínimo apresentou-se como o segundo mais desafiador. Os demais problemas demonstraram taxas de sucesso superiores a 86%.

Tabela 3: Análise de Desempenho por Problema (excluindo Gemini).

	Erros		**Funcionou**	**Tam. Prompt**		
Problema	**Comp.**	**Exec.**	**Correto (%)**	**Primeiro**	**Total**	**Testes**
Arvore Binária	0,00	0,09	100,0	98,0	112,7	30
Grafo Arvore Geradora	0,03	0,17	86,2	149,2	171,9	29
Hash	0,04	0,14	93,1	113,3	182	30
Heap	0,08	0,26	91,2	114,8	144,0	34
Jogo da Velha	0,09	0,22	88,9	104,2	144,7	36
Menor Caminho	0,11	0,21	80,0	127,0	188,5	30
Ordenação Inserção	0,06	0,21	86,7	109,8	179,1	30
Ordenação Seleção	0,09	0,20	86,7	111,6	160,0	30
Turing	0,07	0,31	56,7	109,0	164,7	30

A Tabela 4 apresenta o desempenho comparativo entre 4 alunos de iniciação científica e o docente. O aluno 3 e o docente expandiram seus testes para além dos 9 problemas básicos, incluindo implementações adicionais como jogo resta-um, autômatos, máquina de estados e protocolo de comunicação. Os três primeiros alunos apresentaram desempenho similar na formulação de prompts. O aluno 4 adotou uma abordagem diferenciada, utilizando *prompts* mais concisos, e obteve resultados positivos. Esta estratégia de simplificação dos *prompts* com ajustes demonstrou potencial para melhor convergência, merecendo uma avaliação mais aprofundada em estudos futuros. O docente já possuia experiência prévia no uso de LLMs e avaliou *prompts* mais detalhados.

Tabela 4: Análise de Desempenho por Aluno (excluindo Gemini).

	Erros		**Funcionou**	**Tam. Prompt**		
Aluno	**Comp.**	**Exec.**	**Correto (%)**	**Primeiro**	**Total**	**Testes**
Aluno 1	0,08	0,30	75,5	153,7	196,4	53
Aluno 2	0,05	0,09	79,6	102,4	167,1	54
Aluno 3	0,07	0,39	78,3	125,1	154,6	69
Aluno 4	0,00	0,17	93,0	66,4	101,5	57
Docente	0,17	0,34	90,6	123,6	173,3	64

A Figura 18 mostra o desempenho de gerar uma ferramenta funcional por LLM e por problema. A LLM Claude tem 100% sucesso para árvore geradora, menor caminho e *hash*, a LLM

Copilot para heap, inserção e *hash* e a LLM ChatGPT para jogo da velha e seleção. Todas tiveram sucesso para árvore binária e o pior desempenho foi a máquina de Turing, onde a LLM Claude teve 80% de sucesso nas tentativas.

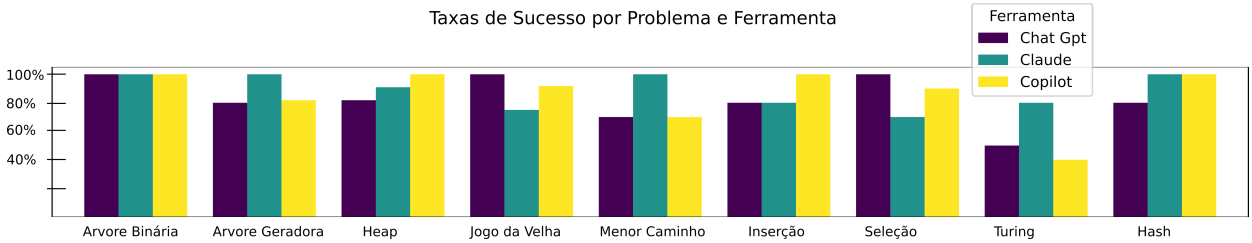

Figura 18: Desempenho das LLMs: ChatGPT, Claude e Copilot por problema.

6.2 Tradução Python para Javascript

Estudos recentes (Pan et al., 2024)demonstram que as LLMs podem ser empregadas na tradução de código, embora o desempenho varie em função do modelo e das linguagens envolvidas. No caso específico da tradução de Python para Java, o desempenho médio do modelo ChatGPT 4.0 foi de aproximadamente 70% de sucesso, avaliado em uma amostra de 614 códigos (Pan et al., 2024). Nesse contexto, elaborou-se o seguinte experimento: a taxa de sucesso na tradução do código de inserção/remoção em uma árvore binária foi avaliada. Cada LLM receberá quatro códigos distintos (gerados originalmente por Claude, ChatGPT, Gemini e Copilot) para traduzir. A pontuação máxima possível por LLM nesta etapa é de 4 pontos, em caso de sucesso total na tradução de todos os códigos. Como foram utilizados códigos de dois estudantes distintos, a pontuação máxima total que um modelo de LLM pode alcançar são 8 pontos.

Tabela 5: Pontuação de funcionalidade das IAs na tradução de Python para JavaScript.

IA Tradutora	Pontuação (Aluno 1)	Pontuação (Aluno 3)	Pontuação Total
Copilot	3.5	2.5	6.0
Claude	4.0	3.0	7.0
Gemini Pro	3.0	1.0	4.0
GPT	3.0	4.0	7.0
Deepseek	4.0	2.0	6.0
Gemini Flash	1.5	1.0	2.5

Os resultados quantitativos da tradução foram analisados com base em dois critérios principais: a funcionalidade do código e a verbosidade (número de linhas). A funcionalidade foi medida por um sistema de pontuação: 1,0 ponto para um código funcional com interface gráfica, 0,5 ponto para um código funcional com interface textual e 0,0 para um código não funcional.

A Tabela 5 consolida a pontuação de cada LLM ao traduzir os códigos dos dois alunos. Os modelos `Claude` e `GPT` se destacaram, alcançando a pontuação total de 7,0, indicando uma alta confiabilidade na geração de aplicações funcionais. Os modelos `Copilot` e `Deepseek` também apresentaram um desempenho sólido, com 6,0 pontos cada. O `Gemini Flash` obteve a menor pontuação, sugerindo menor eficácia nesta tarefa específica. Podemos observar que a ChatGPT e a Claude obtiveram um desempenho de 87% de sucesso, compatível com os experimentos apresentados em (Pan et al., 2024) para a tradução de código Python para Java.

Além da funcionalidade, a verbosidade do código foi avaliada medindo a quantidade média de linhas geradas por cada tradutor, conforme apresentado na Tabela 6. Neste quesito, observa-se uma diferença notável. O modelo `Claude`, apesar de sua alta pontuação de funcionalidade, gerou em média 428.1 linhas de código, uma quantidade significativamente maior que a dos outros modelos. Em contrapartida, `Copilot` e `GPT`, que também tiveram bom desempenho, foram muito mais concisos, com médias de 120,5 e 126.8 linhas, respectivamente.

Tabela 6: Média de linhas de código geradas por cada IA tradutora.

IA Tradutora	Média de Linhas de Código
Copilot	120,5
Claude	428,1
Gemini PRO	185,3
GPT	126,8
DeepSeek	159,1
Gemini Flash	205,1

6.3 Principais Resultados

Como já mencionado, este trabalho apresenta uma avaliação exploratória de LLMs comerciais para o desenvolvimento de ferramentas educacionais interativas voltadas ao ensino de estruturas de dados e algoritmos. A Tabela 7 sintetiza as principais contribuições organizadas em categorias.

Tabela 7: Síntese dos Principais Resultados.

Categoria	Descrição
Dataset **Educacional**	Criação de um conjunto de dados para ferramentas visuais para problemas clássicos de computação com simulação interativa, indo além de trabalhos existentes focados em diagramas estruturados.
Documentação do Processo Exploratório	Documentação sistemática do processo exploratório, incluindo formulação de *prompts*, análise de efetividade e validação de ferramentas geradas por quatro LLMs comerciais (ChatGPT, Claude, Copilot e Gemini).
Avaliação Python e Javascript	Exemplos nas duas linguagens mais populares: Python (ambiente Colab) e JavaScript (execução nativa em navegadores ou Colab).
Recurso de Edição	Extensão da análise para geração de interfaces mais interativas com recursos de editores gráficos.
Autoavaliação	Exemplos de análise da capacidade de autoavaliação utilizando códigos gerados por uma LLM e avaliados por outra, incentivando reflexão sobre qualidade do código.
DSLs Educacionais	Exploração da geração de interpretadores para linguagens de domínio específico (DSL) voltadas ao ensino, com exemplos em grafos e inteligência artificial, área de forte potencial das LLMs (Joel et al., 2024).
Material Avaliativo	Exemplo de geração de questionários executáveis no Google Colab e interfaces adaptadas para dispositivos móveis, visando fixação de conceitos de forma interativa.

7 Conclusões

As LLMs estabeleceram-se como uma tecnologia permanente, mesmo considerando o elevado custo de construção e manutenção dos *datacenters* (Khowaja et al., 2024) e as incertezas quanto à disponibilidade futura das opções gratuitas a curto prazo. Este trabalho estendido avaliou o potencial das LLMs para criação de recursos educacionais em sete eixos: visualização gráfica e interatividade em Python, em JavaScript e tradução de Python para JavaScrit, edição gráfica, linguagens de domínio específico, avaliação automatizada de código e geração de questionários.

Os resultados demonstraram que é possível, em questão de minutos, gerar interfaces interativas. A robustez do sistema foi evidenciada frente a variações nos *prompts*, alcançando taxas de sucesso superiores a 80% com descrições de aproximadamente 100 palavras ou 6-8 linhas para cada ferramenta gerada. Os códigos gerados são concisos, contendo em média 100 linhas, incluindo comentários. A geração de editores gráficos e a linguagem de domínio específico permitem que, após uma única geração inicial pela LLM, professores e alunos utilizem camadas de abstração para produzir uma diversidade de exemplos educacionais além dos exemplos aleatórios.

A avaliação comparativa de seis diferentes LLMs (Gemini, ChatGPT, Copilot, Claude, Deepseek e Gemini Pro) na tradução de Python para JavaScript, validou a robustez da abordagem e demonstrou capacidades similares entre os modelos avaliados. A documentação dos *prompts* e códigos, estabelece um *dataset* de geração de 382 ferramentas funcionais na área de ferramentas didáticas computacionais com interface.

Embora a construção incremental das ferramentas com *prompts* simples tenha se mostrado eficaz, observamos algumas instabilidades, o que é esperado devido à natureza emergente da tecnologia. No entanto, com uma amostragem de mais de 450 ensaios, os resultados evidenciam não apenas o potencial individual das LLMs, mas também sua capacidade de criar ecossistemas educacionais autossustentáveis, gerando mais de 380 ferramentas com interfaces funcionais.

As LLMs também podem ser utilizadas para melhorar a produção de material didático em domínios específicos juntamente com o Google Colab, como, por exemplo, na área de arquitetura de computadores (Canesche et al., 2021; Ferreira, Canesche et al., 2024; Ferreira, Sabino et al., 2024; Figueiredo et al., 2024), consolidando-se seu papel na modernização do ensino, através da criação de camadas de abstração que democratizam a geração de conteúdo interativo.

Assim, do ponto de vista educacional, os resultados obtidos reforçam que as LLMs oferecem um potencial significativo para ampliar a criação e o acesso a ferramentas didáticas interativas, possibilitando que docentes e discentes desenvolvam recursos de forma mais ágil e diversificada. Entretanto, sua integração ao ensino exige uma abordagem crítica e informada. Primeiramente, é necessário fomentar a alfabetização em IA entre professores de outras áreas (que não de IA) e de estudantes também de outras áreas ou em etapas iniciais de cursos relacionados à Computação, abrangendo tanto o funcionamento básico dos modelos quanto seus impactos pedagógicos, considerando o uso responsável de sistemas algorítmicos (Floridi & Cowls, 2022; Mitchell, 2019).

Quanto às limitações observadas, verificou-se que os modelos apresentam variações significativas em precisão, estabilidade e consistência. Além disso, erros conceituais e fenômenos de alucinação, amplamente analisados por (Bender et al., 2021), reforçam a necessidade de validação docente contínua do material produzido. Tais limitações são agravadas pela natureza opaca dos modelos, dificultando a compreensão dos processos subjacentes à geração das respostas. Nossa

proposta é sempre documentar os códigos já validados e tê-los como referência, evitando regenerá-los desnecessariamente e promovendo sua reutilização.

Em relação aos desafios éticos, questões como privacidade, autoria e vieses algorítmicos, tornam-se centrais. Estudos como (Gebru et al., 2021) alertam para o risco de reprodução de desigualdades sistêmicas e para a falta de transparência na cadeia de dados e nos processos de modelagem. Assim, o uso responsável de LLMs no contexto educacional requer diretrizes institucionais claras, políticas de governança e formação docente contínua, de modo a assegurar que a produção de materiais pedagógicos respeite princípios de equidade, justiça e confiabilidade.

Dessa forma, embora este estudo evidencie o potencial das LLMs para apoiar a criação de ferramentas educacionais na área de Computação, sua utilização sustentável depende da articulação entre práticas pedagógicas robustas, formação crítica em IA e mecanismos de governança ética que orientem seu uso seguro e adequado dessas tecnologias no ensino.

O material completo está disponível em repositório público (UFV, 2025), servindo como fonte de dados para análises posteriores e estabelecendo um *framework* inicial para a comunidade. Pretende-se investigar também a similaridade entre os códigos gerados das ferramentas e expandir as linguagens de domínio específico para outras áreas da computação.

Como desdobramento deste trabalho, planejamos conduzir um estudo em contexto educacional com estudantes, de modo a avaliar empiricamente o potencial pedagógico das ferramentas geradas pelas LLMs. Esse estudo será estruturado no formato pré-teste/intervenção/pós-teste, no qual os participantes utilizarão algumas das ferramentas desenvolvidas, como as visualizações de árvores binárias e tabelas *hash*, durante atividades práticas supervisionadas, permitindo examinar seu impacto na compreensão conceitual, na motivação e na experiência de uso. Complementarmente, será realizada uma avaliação pedagógica com docentes da área, que analisarão a clareza, a precisão conceitual, a utilidade instrucional e a adequação das ferramentas para uso em sala de aula, contribuindo com *feedback* técnico. Tanto estudantes quanto professores responderão instrumentos como o System Usability Scale (SUS), questionários de percepção de utilidade e experiência, além de fornecerem comentários abertos que serão analisados qualitativamente. A realização desse piloto permitirá não apenas verificar a efetividade das ferramentas no apoio ao ensino, mas também identificar limitações, ajustes necessários e diretrizes de uso pedagógico, ampliando o alcance e fortalecendo a contribuição deste estudo para a área de Educação em Computação.

Por fim, reconhecemos que o caráter exploratório desta investigação, centrado na geração e análise das ferramentas e ainda sem validação pedagógica com usuários reais, constitui uma limitação do estudo, especialmente quando comparado a abordagens empíricas mais consolidadas na área. Ainda assim, o trabalho se configura como um relato de pesquisa em desenvolvimento e exploração tecnológica, oferecendo contribuições iniciais que sistematizam processos, evidenciam potencialidades e orientam futuras investigações empíricas com estudantes e docentes.

Agradecimentos

Apoio financeiro da Bolsa de Iniciação Científica da FAPEMIG e do CNPq programa Institucional, Bolsa PIBEN Funarbe/Pro-Reitoria de Ensino UFV, FAPEMIG APQ-01577-22, CNPq e Centro de Ciência Exatas e Tecnológicas da UFV.

Referências

Al-Shetairy, M., Hindy, H., Khattab, D., & Aref, M. M. (2024). Transformers Utilization in Chart Understanding: A Review of Recent Advances & Future Trends. *arXiv preprint arXiv:2410.13883*. https://doi.org/10.48550/arXiv.2410.13883 [GS Search].

Barbosa, L. L., Couto, C. M. S., & Terra, R. (2016). PortuCol: uma pseudo linguagem inspirada em C ANSI para o Ensino de Lógica de Programação e Algoritmos. *Workshop sobre Educação em Computação (WEI)*, 2343–2352. https://doi.org/10.5753/wei.2016.9678 [GS Search].

Bender, E. M., Gebru, T., McMillan-Major, A., & Mitchell, M. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. https://doi.org/10.1145/3442188.3445922 [GS Search].

Cai, Y., Mao, S., Wu, W., Wang, Z., Liang, Y., Ge, T., Wu, C., You, W., Song, T., Xia, Y., Duan, N., & Wei, F. (2023). Low-code LLM: Graphical User Interface over Large Language Models. *arXiv preprint arXiv:2304.08103*. https://doi.org/10.18653/v1/2024.naacl-demo.2 [GS Search].

Canesche, M., Bragança, L., Vilela Neto, O. P., Nacif, J. A., & Ferreira, R. (2021). Google Colab CAD4U: Hands-on cloud laboratories for digital design. *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401151 [GS Search].

Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2023). Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. *arXiv preprint arXiv:2310.14735*. https://doi.org/10.1016/j.patter.2025.101260 [GS Search].

Chen, C., Sonnert, G., Sadler, P. M., & Malan, D. J. (2020). Computational thinking and assignment resubmission predict persistence in a computer science MOOC. *Journal of Computer Assisted Learning*, *36*(5), 581–594. https://doi.org/10.1111/jcal.12427 [GS Search].

Chen, L., Guo, Q., Jia, H., Zeng, Z., Wang, X., Xu, Y., Wu, J., Wang, Y., Gao, Q., Wang, J., Ye, W., & Zhang, S. (2024). A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498*. https://doi.org/10.48550/arXiv.2408.16498 [GS Search].

Chu, Z., Wang, S., Xie, J., Zhu, T., Yan, Y., Ye, J., Zhong, A., Hu, X., Liang, J., Yu, P. S., & Wen, Q. (2025a). LLM Agents for Education: Advances and Applications. *arXiv preprint arXiv:2503.11733*. [GS Search].

Chu, Z., Wang, S., Xie, J., Zhu, T., Yan, Y., Ye, J., Zhong, A., Hu, X., Liang, J., Yu, P. S., & Wen, Q. (2025b). LLM agents for education: Advances and applications. *arXiv preprint arXiv:2503.11733*. https://doi.org/10.18653/v1/2025.findings-emnlp.743 [GS Search].

Coura, P., Freitas, I., Costa, H., Nacif, J., & Ferreira, R. (2025). Desmistificando o Ensino de Inteligência Artificial e Aprendizado de Máquina. *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, 25–27. https://doi.org/10.5753/educomp_estendido.2025.6578 [GS Search].

Del, M., & Fishel, M. (2022). True detective: a deep abductive reasoning benchmark undoable for GPT-3 and challenging for GPT-4. *arXiv preprint arXiv:2212.10114*. https://doi.org/10.18653/v1/2023.starsem-1.28 [GS Search].

Elon University. (2025, março). Survey: 52% of U.S. adults now use AI large language models like ChatGPT. Disponível em: [Link]. Acessado em: 25 de março de 2026.

Ferreira, R., Canesche, M., Jamieson, P., Vilela Neto, O. P., & Nacif, J. A. (2024). Examples and tutorials on using Google Colab and Gradio to create online interactive student-learning modules. *Computer Applications in Engineering Education*, e22729. https://doi.org/10.1002/cae.22729 [GS Search].

Ferreira, R., Sabino, C., Canesche, M., Vilela Neto, O. P., & Nacif, J. A. (2024). AIoT tool integration for enriching teaching resources and monitoring student engagement. *Internet of Things*, *26*, 101045. https://doi.org/10.1016/j.iot.2023.101045 [GS Search].

Figueiredo, G. A. R., Souza, E. S., Rodrigues, J. H. F., Nacif, J. A., & Ferreira, R. (2024). Desenvolvendo Ferramentas para Ensino de RISC-V com Python, Verilog, Matplotlib, SVG e ChatGPT. *International Journal of Computer Architecture Education*, *13*(1), 43–52. https://doi.org/10.5753/ijcae.2024.5343 [GS Search].

Floridi, L., & Cowls, J. (2022). A unified framework of five principles for AI in society. *Machine learning and the city: Applications in architecture and urban design*, 535–545. https://doi.org/10.1162/99608f92.8cd550d1 [GS Search].

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Iii, H. D., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM*, *64*(12), 86–92. https://doi.org/10.1145/3458723 [GS Search].

Hu, B., Zhu, J., Pei, Y., & Gu, X. (2025). Exploring the potential of LLM to enhance teaching plans through teaching simulation. *npj Science of Learning*, *12*(1), 1–12. https://doi.org/10.1038/s41539-025-00300-x [GS Search].

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). Swebench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*. https://doi.org/10.48550/arXiv.2310.06770 [GS Search].

Joel, S., Wu, J. J., & Fard, F. H. (2024). A survey on LLM-based code generation for low-resource and domain-specific programming languages. *arXiv preprint arXiv:2410.03981*. https://doi.org/10.1145/3770084 [GS Search].

Joshi, S. (2025). Open-Source vs. Commercial Coding Assistants: A 2025 Comparison of DeepSeek R1, Qwen 2.5 and Claude 3.7. *International Journal of Computer Applications Technology and Research*, *14*(09), 6–18. [GS Search].

Julio, J. P. F., Campano Junior, M. M., Aylon, L. B. R., Fonseca, K. O., & Emmendörfer, L. R. (2024). Jogos educativos para Estruturas de Dados: Um Mapeamento Sistemático. *Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*, 1186–1199. https://doi.org/10.5753/sbgames.2024.240933 [GS Search].

Kazemitabaar, M., Hou, X., Henley, A., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. *Proceedings of the 23rd Koli calling international conference on computing education research*, 1–12. https://doi.org/10.1145/3631802.3631806 [GS Search].

Khowaja, S. A., Khuwaja, P., Dev, K., Wang, W., & Nkenyereye, L. (2024). ChatGPT needs spade (sustainability, privacy, digital divide, and ethics) evaluation: A review. *Cognitive Computation*, 1–23. https://doi.org/10.1007/s12559-024-10285-1 [GS Search].

Kiesler, N., & Schiffner, D. (2023). Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. *arXiv preprint arXiv:2308.08572*. https://doi.org/10.48550/arXiv.2308.08572 [GS Search].

Kumar, K. P., Reddy, G. A., Vignesh, D., Pavan, A., Vamsi, T., Rishi, K., & Kumar, K. S. (2025). Automated Question Paper Generator Using LLM. *International Journal of Research and*

Innovation in Applied Science, *10*(4), 266–275. https://doi.org/10.51584/IJRIAS.2025.10040020 [GS Search].

Lisboa, M. O., Costa, H., Coura, P., Freitas, I., Villela, M. L. B., & Ferreira, R. (2025). Modelos Generativos de Linguagem na Construção de Ferramentas de Ensino de Computação com Interface Gráfica. *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, 639–650. https://doi.org/10.5753/educomp.2025.4927 [GS Search].

Liu, J., Liu, A., Lu, X., Welleck, S., West, P., Bras, R. L., Choi, Y., & Hajishirzi, H. (2021). Generated knowledge prompting for commonsense reasoning. *arXiv preprint arXiv:2110.08387*. https://doi.org/10.18653/v1/2022.acl-long.225 [GS Search].

Logan IV, R. L., Balažević, I., Wallace, E., Petroni, F., Singh, S., & Riedel, S. (2021). Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint arXiv:2106.13353*. https://doi.org/10.18653/v1/2022.findings-acl.222 [GS Search].

Lyu, W., Wang, Y., Sun, Y., & Zhang, Y. (2025). Will Your Next Pair Programming Partner Be Human? An Empirical Evaluation of Generative AI as a Collaborative Teammate in a Semester-Long Classroom Setting. *Proceedings of the Twelfth ACM Conference on Learning@ Scale*, 83–94. https://doi.org/10.1145/3698205.3729544 [GS Search].

Ma, X., Liu, Q., Jiang, D., Zhang, G., Ma, Z., & Chen, W. (2025). General-reasoner: Advancing llm reasoning across all domains. *arXiv preprint arXiv:2505.14652*. https://doi.org/10.48550/arXiv.2505.14652 [GS Search].

Martins, F. L. B., Oliveira, A. C. A., Vasconcelos, D. R., & Menezes, M. V. (2025). Avaliando a habilidade do ChatGPT de realizar provas de Dedução Natural em Lógica Proposicional e Lógica de Predicados. *Revista Brasileira de Informática na Educação*, *33*, 244–278. https://doi.org/10.5753/rbie.2025.4500 [GS Search].

Martins, R. M., von Wangenheim, C. G., Rauber, M. F., & Hauck, J. C. (2024). Machine learning for all!—introducing machine learning in middle and high school. *International Journal of Artificial Intelligence in Education*, *34*(2), 185–223. https://doi.org/10.1007/s40593-022-00325-y [GS Search].

Mitchell, M. (2019). *Artificial intelligence: A guide for thinking humans*. Penguin UK. [GS Search].

Mutanga, M. B., Msane, J., Mndaweni, T. N., Hlongwane, B. B., & Ngcobo, N. Z. (2025). Exploring the Impact of LLM Prompting on Students' Learning. *Trends in Higher Education*, *4*(3), 31. https://doi.org/10.3390/higheredu4030031 [GS Search].

Pan, R., Ibrahimzada, A. R., Krishna, R., Sankar, D., Wassi, L. P., Merler, M., Sobolev, B., Pavuluri, R., Sinha, S., & Jabbarvand, R. (2024). Lost in translation: A study of bugs introduced by large language models while translating code. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13. https://doi.org/https://doi.org/10.1145/3597503.3639226 [GS Search].

Park, J., Teo, T. W., Teo, A., Chang, J., Huang, J. S., & Koo, S. (2023). Integrating artificial intelligence into science lessons: Teachers' experiences and views. *International Journal of STEM Education*, *10*(1), 61. https://doi.org/10.1186/s40594-023-00454-3 [GS Search].

Pereira Filho, L. C., Souza, T. P. C., & Paula, L. B. (2025). Análise das Respostas de LLMs em Relação ao Conteúdo Introdutório de Programação: um Comparativo entre o ChatGPT e o Gemini. *Revista Brasileira de Informática na Educação*, *33*, 722–747. https://doi.org/10.5753/rbie.2025.4477 [GS Search].

Phogat, R., Arora, D., Mehra, P. S., Sharma, J., & Chawla, D. (2025). A Comparative Study of Large Language Models: ChatGPT, DeepSeek, Claude and Qwen. *2025 3rd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT)*, 609–613. https://doi.org/10.1109/DICCT64131.2025.10986449 [GS Search].

Russo, F. A. I., Rabelo e Sant'Anna, N., & Imai, R. H. (2023). Relato de experiência educacional com o uso de inteligências artificiais sintetizadoras de imagens: debate sobre avanços recentes e possibilidades em síntese criativa. *Revista Brasileira de Informática na Educação*, *31*, 814–828. https://doi.org/10.5753/rbie.2023.2914 [GS Search].

Sato, Y., Suzuki, A., & Mineshima, K. (2024). Building a Large Dataset of Human-Generated Captions for Science Diagrams. *International Conference on Theory and Application of Diagrams*, 393–401. https://doi.org/10.1007/978-3-031-71291-3_32 [GS Search].

UFV. (2025). Universidade Federal de Viçosa - Material Complementar. [Link].

Vasconcelos, V. R. C., & Frota, D. A. (2025). IA como Aliada da Criatividade Humana: O Desenvolvimento do Aplicativo "Terra e Universo" com Auxílio do ChatGPT. *Revista Brasileira de Informática na Educação*, *33*, 451–471. https://doi.org/10.5753/rbie.2025.5204 [GS Search].

Verhalen, L. E. C., Castro, M. M. M., & Maciel, C. (2025). Percepções e Ferramentas sobre Recriação Digital de Educadores por meio de Inteligência Artificial: Uma Revisão Sistemática de Literatura. *Revista Brasileira de Informática na Educação*, *33*, 565–582. https://doi.org/10.5753/rbie.2025.5279 [GS Search].

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, *35*, 24824–24837. [GS Search].

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*. https://doi.org/10.48550/ARXIV.2302.11382 [GS Search].

Xu, X., Tao, C., Shen, T., Xu, C., Xu, H., Long, G., Lou, J.-g., & Ma, S. (2023). Re-reading improves reasoning in language models. *arXiv preprint arXiv:2309.06275*. [GS Search].

Yan, L., Sha, L., Zhao, L., Li, Y., Martinez-Maldonado, R., Chen, G., Li, X., Jin, Y., & Gašević, D. (2024). Practical and ethical challenges of large language models in education: A systematic scoping review. *British Journal of Educational Technology*, *55*(1), 90–112. https://doi.org/10.1111/bjet.13370 [GS Search].

Yan, Y.-M., Chen, C.-Q., Hu, Y.-B., & Ye, X.-D. (2025). LLM-based collaborative programming: impact on students' computational thinking and self-efficacy. *Humanities and Social Sciences Communications*, *12*(1), 1–12. https://doi.org/10.1057/s41599-025-04471-1 [GS Search].

Yang, Z., Li, L., Wang, J., Lin, K., Azarnasab, E., Ahmed, F., Liu, Z., Liu, C., Zeng, M., & Wang, L. (2023). MM-REACT: Prompting ChatGPT for Multimodal Reasoning and Action. *arXiv preprint arXiv:2303.11381*. https://doi.org/10.48550/arXiv.2303.11381 [GS Search].

Yang, Z., & Zhu, Z. (2024). Heuristic question sequence generation based on retrieval augmentation. *Education and Lifelong Development Research*. https://doi.org/10.46690/elder.2024.02.03 [GS Search].

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2024). Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, *36*. [GS Search].

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*. https://doi.org/10.48550/ARXIV.2210.03629 [GS Search].

Yuan, Y., Li, Z., & Zhao, B. (2025). A survey of multimodal learning: Methods, applications, and future. *ACM Computing Surveys*, *57*(7), 1–34. https://doi.org/10.1145/3713070 [GS Search].

Zala, A., Lin, H., Cho, J., & Bansal, M. (2023). DiagrammerGPT: Generating Open-Domain, Open-Platform Diagrams via LLM Planning. *arXiv preprint arXiv:2310.12128*. https://doi.org/10.48550/arXiv.2310.12128 [GS Search].

Zanini, A. S., & Raabe, A. L. A. (2012). Análise dos enunciados utilizados nos problemas de programação introdutória em cursos de Ciência da Computação no Brasil. *Workshop sobre Educação em Computação (WEI)*, 11–20. [GS Search].

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., & Chi, E. (2022). Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*. https://doi.org/10.48550/arXiv.2205.10625 [GS Search].

Material Suplementar

Este artigo disponibiliza material complementar utilizando um Google Colab como índice (UFV, 2025). O material está organizado em nove seções principais: animações, resumos das métricas, experimentos com *prompts* simples, experimentos com *prompts* detalhados, geração com JavaScript, tradução de Python para JavaScript, edição gráfica e linguagens de domínio específico para grafos, análise de código com LLMs e questionário com JSON. A primeira seção do material suplementtar contém 53 animações no formato GIF, onde podemos observar de forma sucinta um exemplo ilustrativo das ferramentas gráficas produzida. A segunda seção apresenta ferramentas para exploração dos dados gerados usando recursos interativos que permitem selecionar o problema, a ferramenta, a técnica, etc. A Figura 19 ilustra os resultados obtidos para o exemplo da heap usando as quatro ferramentas com o tamanho médio dos *prompts* em palavras, taxa de sucesso dos testes e o número médio de linhas de código geradas.

Tabela para o Problema: Heap

Ferramenta	Número de Ocorrências	Funcionou Corretamente	Média de Palavras nos Prompts	Média de Linhas de Código
Chat Gpt	11	9	150.2	85.545455
Claude	11	10	157.5	121.272727
Copilot	12	12	125.9	70.250000
Gemini	7	5	133.0	82.428571

Figura 19: Tabelas Interativas para Exploração dos Dados dos Experimentos.

A terceira e quarta seção são organizadas por problema e por usuário. Para cada problema, existe uma sub-seção para usuário, onde uma tabela resumida dos experimentos com número de tentativas ou *shots*(T), erros de compilação e execução, tamanho inicial e total em palavras dos prompts, linhas de código geradas para a última implementação, funcionalidade e qual técnica de *prompt* que foi utilizada. Toda tabela tem um link para um Google Colab que contém o experimento que foi executado e pode ser reproduzido, com os *prompts* usados e os resultados obtidos com todos os seus detalhes. São 106 sub-seções, onde cada uma delas contém uma tabela e o detalhamento dos resultados.

Tabela 8: Exemplo de Tabela para o *hash* e programador 1 com a comparação entre ferramentas.

LMM	**Erros Comp.**	T	**Erros Exec.**	1^0 **Prompt**	**Todos os Prompts**	**Linhas Código**	**Funcionalidade**	**Técnica de Prompt**
Chat	0	2	1	30	63	153	Correta	Simples
Claude	0	1	0	30	30	205	Correta	Simples
Copilot	0	1	0	30	30	84	Correta	Simples
Gemini	0	1	0	30	30	97	Correta	Simples

A quinta seção mostra a geração para os mesmos problemas usando como alvo a linguagem JavaScript. A sexta seção avalia a capacidade de seis ferramentas de fazer a tradução dos códigos gerados em Python para JavaScript, sem auxílio da descrição do *prompt* que gerou o código pelas quatro LLMs iniciais no exemplo da interface para ensino de árvores binárias. A sétima seção apresenta dois resultados para o ensino de grafos: primeiro, a ferramenta de edição; depois, as ferramentas com linguagens de domínio específico. A oitava seção apresenta o uso das LLMs para análise do código Python gerado pelas próprias LLMs. Finalmente, a nona seção do material complementar mostra mais um exemplo de uso de LLMs para questionários, buscando uma interface compatível com dispositivos móveis como celulares e um formato simples para construção de ferramentas de autoavaliação.