Geração Automática de Questões de Programação Usando LLM: Um Relato de Experiência

Abner Santana[1], **Francisco Genivan Silva**[1 2],
Júlio Dantas[1], **Jadson Souza**[1], **Eduardo Aranha**[1]

[1]Programa de Pós-Graduação em Sistemas Computacionais – PPgSC (UFRN)
Natal - Rio Grande do Norte - Brasil

[2]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte - IFRN
Natal - Rio Grande do Norte - Brasil

abnersant10@gmail.com, genivan.silva@ifrn.edu.br,
{dantas.j.c.s,jadsonlukas}@gmail.com, eduardo.aranha@ufrn.br

Abstract. *This paper reports an experience of automatically generating 180 introductory programming questions using large language models (LLMs). The methodology was based on iterative cycles of prompt engineering, involving the use of structured templates, guided examples (few-shot prompting), and automated refinement (self-refinement). The approach aimed to ensure clarity, completeness, and pedagogical alignment in the generated questions. Results indicate that this type of strategy proved effective and replicable, contributing to scalable educational content production supported by artificial intelligence. Furthermore, several lessons learned are presented to empower educators and researchers in applying these techniques.*

Resumo. *Este artigo relata uma experiência de geração automática de 180 questões de programação introdutória usando modelos de linguagem de grande porte (LLMs). A metodologia foi baseada em ciclos iterativos de engenharia de prompt, envolvendo o uso de templates estruturados, exemplos guiados (few-shot prompting) e refinamento automatizado (self-refinement). A abordagem buscou garantir clareza, completude e alinhamento pedagógico nas questões geradas. Os resultados mostram que esse tipo de estratégia demonstrou-se eficaz e replicável, contribuindo para a produção escalável de conteúdo educacional com apoio de inteligência artificial. Além disso, diversas lições aprendidas são apresentadas, visando empoderar professores e pesquisadores com o uso dessas técnicas.*

1. Introdução

A resolução de exercícios desempenha um papel importante no aprendizado dos alunos, principalmente quando se está aprendendo programação. Para que isso ocorra adequadamente, é necessária a criação de bases de questões de programação bem elaboradas, categorizadas por assunto e níveis de dificuldade, e contextualizadas com problemas observados no dia a dia da sociedade [Thalheimer 2003]. Isto é especialmente importante ao se utilizar teorias e práticas pedagógicas como *Mastery Learning* [Carneiro et al. 2022]. No entanto, a criação manual de questões requer um certo esforço e a disponibilidade de especialistas com experiência na tarefa [Kurdi et al. 2020].

Diante dessas limitações, pesquisadores têm buscado abordagens automatizadas para tornar a geração de questões mais eficiente e acessível. A automação desse processo não apenas reduz a carga de trabalho dos educadores, permitindo que se concentrem em outras atividades pedagógicas, mas também contribui para a melhoria da aprendizagem, oferecendo questões adaptadas e diversificadas de maneira escalável. Tradicionalmente, os sistemas de geração automática de questões baseiam-se em regras predefinidas, análise sintática e semântica, templates estruturados e modelos estatísticos mais simples. Esses métodos, embora úteis, apresentam limitações, como a necessidade de intervenção manual para ajustes e a dificuldade de adaptação a diferentes contextos e estilos de aprendizado.

Nos últimos anos, pesquisas têm avançado para o uso de grandes modelos de linguagem (LLMs) como uma alternativa mais flexível para essa tarefa. Esses modelos são capazes de gerar questões variadas e contextualmente relevantes a partir de grandes volumes de dados. No entanto, apesar das vantagens destacadas na literatura, como a capacidade de personalização e adaptação ao conteúdo, essas aplicações ainda enfrentam desafios recorrentes. No estudo de [Meißner et al. 2024], por exemplo, foram necessárias várias reformulações de prompts e um processo iterativo manual para a realização da tarefa. Ainda assim, os autores relataram que alguns modelos falharam em seguir padrões de design da programação orientada a objetos, em adequar o nível de dificuldade ao especificado no prompt e em identificar e classificar erros no código. Além disso, a efetividade pedagógica das questões geradas também foi pouco investigada [Faraby et al. 2024].

Nesse sentido, este artigo relata uma experiência realizada na aplicação de LLMs e de técnicas de engenharia de prompts para a geração de questões de programação para uma disciplina de programação introdutória de um curso de graduação da UFRN. São apresentadas diferentes tentativas de geração de questões, sendo relatada para cada uma delas a estratégia utilizada, bem como os benefícios e limitações observados. Desta forma, este artigo contribui para que professores e pesquisadores possam se inspirar no que foi realizado e, a partir das lições aprendidas relatadas aqui, evoluir e promover suas próprias gerações de questões de programação.

2. Trabalhos relacionados

Pesquisas anteriores já exploraram o uso de modelos de linguagem de grande porte (LLMs) na geração automática de questões, analisando seus benefícios e desafios. O estudo de [Chan et al. 2023] investiga especificamente a capacidade do Chat-GPT na formulação de perguntas abertas a partir de trechos de texto, utilizando técnicas de prompt engineering para otimizar os resultados. Um dos aspectos mais relevantes desse estudo é a criação de um avaliador automatizado baseado em GPT, projetado para atribuir notas às perguntas geradas. Esse avaliador é comparado ao julgamento de avaliadores humanos, permitindo uma análise sobre a precisão e a confiabilidade do modelo na tarefa de avaliação da qualidade das questões produzidas.

O trabalho de [Meißner et al. 2024] investiga a geração automatizada de exercícios de programação no ensino superior, com foco em perguntas abertas. Os autores comparam o desempenho de diferentes LLMs, incluindo ChatGPT, Bing AI Chat e Google Bard, na criação de exercícios de programação voltados para cursos de nível iniciante a intermediário. Para avaliar a qualidade das questões geradas, cada exercício foi

revisado manualmente, e os pesquisadores registraram o número de iterações necessárias para alcançar um nível satisfatório de qualidade. A análise revelou que, embora os LLMs apresentem um grande potencial para automatizar a criação de exercícios de programação, garantindo coerência e alinhamento pedagógico, a supervisão humana continua sendo essencial. O envolvimento de especialistas desempenha um papel crucial na adaptação dos exercícios ao nível de dificuldade adequado, na correção de eventuais falhas e no aprimoramento da clareza e precisão das questões geradas. Os resultados destacam a importância da interação entre inteligência artificial e intervenção humana para garantir que os exercícios automatizados sejam não apenas tecnicamente corretos, mas também eficazes do ponto de vista didático, promovendo um aprendizado estruturado e alinhado aos objetivos educacionais.

Os autores de [Niu and Xue 2023] exploram o uso do Chat-GPT na elaboração de exercícios de múltipla escolha para um curso de sistemas operacionais. A abordagem adotada combina a geração automática de questões com a aplicação do modelo de Rasch, uma técnica estatística utilizada para avaliar o nível de dificuldade dos exercícios e ajustá-los ao nível de habilidade dos estudantes. Ao integrar a inteligência artificial ao modelo de Rasch, os pesquisadores buscaram otimizar a correspondência entre a complexidade das questões e o perfil do aluno, permitindo um ajuste dinâmico na dificuldade dos exercícios. Os resultados do estudo indicam que essa abordagem tem um grande potencial para aprimorar tanto a eficiência quanto a eficácia do processo de criação de exercícios, reduzindo o esforço manual dos educadores sem comprometer a qualidade pedagógica. Além disso, a personalização proporcionada pela técnica pode contribuir para uma experiência de aprendizado mais adaptativa, incentivando o engajamento dos estudantes e promovendo um ensino mais alinhado às suas necessidades individuais.

Também no contexto de questões objetivas, [Doughty et al. 2024] investigam a criação automatizada de perguntas para cursos de programação em Python no ensino superior. A abordagem adotada baseia-se no currículo e nos objetivos de aprendizagem de cada módulo para orientar a geração das questões, garantindo que elas sejam relevantes e alinhadas ao conteúdo ensinado. Para avaliar a qualidade das questões geradas, os pesquisadores empregaram uma rubrica com critérios específicos, analisando aspectos como a presença de informações suficientes para responder à pergunta, a existência de uma única alternativa correta, a originalidade das opções de resposta, a ausência de escolhas obviamente erradas, a exatidão do código apresentado e a correspondência da questão com os objetivos de aprendizagem. Os resultados indicaram que o modelo GPT-4 demonstrou uma capacidade notável de produzir questões bem formuladas, utilizando uma linguagem clara e objetiva. Além disso, as perguntas geradas apresentaram uma única resposta correta e distratores (alternativas incorretas) de alta qualidade, contribuindo para a construção de testes mais desafiadores e eficazes do ponto de vista didático. Esses achados reforçam o potencial da inteligência artificial para otimizar a criação de avaliações no ensino de programação, proporcionando maior eficiência para os educadores e uma experiência de aprendizado mais estruturada para os estudantes.

Uma característica comum a maior parte dos trabalhos que discutem o uso de aplicações como o Chat-GPT na geração de questões é usar uma única chamada para gerar as questões: isto é, embora adotem diferentes técnicas, como o few-shoot e a chain-of-thought, a maior parte das aplicações se baseiam em um longo e detalhado prompt

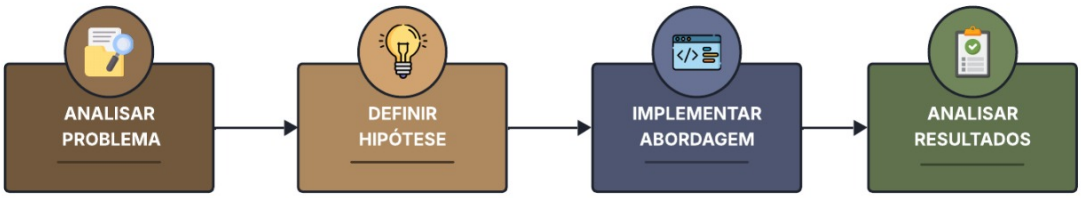

Figure 1. Metodologia adotada na geração de questões.

na geração, como em [Chan et al. 2023], às vezes somando a isso a avaliação humana de cada questão, como em [Meißner et al. 2024]. Sendo assim, este trabalho avança em relação ao estado corrente da literatura quando propõe, em três formatos diferentes, chamadas aos assistentes de inteligência artificial em camadas, com cada uma tratando de um aspecto diferente na geração - a ser detalhado adiante. Além disso, exclui também do fluxo de atividades a avaliação humana, buscando ser verdadeiramente automático.

3. Metodologia

A experiência relatada neste artigo utilizou uma abordagem metodológica estruturada em ciclos iterativos, com o uso de engenharia de prompts e o GPT-4, visando otimizar a geração automatizada de questões para uma disciplina de graduação de programação introdutória. Como ilustrado na Figura 1, a metodologia baseia-se em quatro etapas sequenciais e interdependentes:

- **Análise do Problema:** antes de cada ciclo, incluindo o inicial, foram analisados problemas ou limitações identificados nas questões geradas anteriormente, exceto no primeiro ciclo, que analisou desafios gerais da geração de questões. Esta análise envolveu identificar padrões recorrentes, erros sistemáticos e lacunas pedagógicas.
- **Formulação de Hipóteses:** a partir da análise do problema, foram elaboradas hipóteses identificando oportunidades de melhoria que poderiam ser exploradas no ciclo atual.
- **Definição e Implementação da Abordagem:** a partir da validação interna das hipóteses, ajustes são incorporados no pipeline de geração para mitigar falhas. Para isso, técnicas de engenharia de *prompt* eram refinadas e adaptadas a cada cenário.
- **Análise dos Resultados:** Após cada ciclo de geração, realizou-se uma análise rigorosa das questões geradas, conduzida pelos pesquisadores em conjunto com professores especialistas. Esta etapa buscou verificar se as estratégias adotadas reduziram ou solucionaram os problemas identificados anteriormente.

A abordagem adotada alinha-se assim ao paradigma do Design Science Research [Hevner et al. 2004], no qual a geração de questões atua como um artefato projetado para resolver problemas pedagógicos identificados, enquanto os ciclos iterativos de refinamento garantem a validação progressiva da solução proposta. Esta configuração cíclica permitiu refinamentos incrementais, ajustando-se às lacunas identificadas em cada iteração, as quais são detalhadas na seção a seguir.

4. Ciclos de Geração de Questões

Esta seção descreve o processo evolutivo da estratégia metodológica ao longo de três ciclos iterativos, detalhando como cada iteração incorporou ajustes baseados em lacunas identificadas nas fases anteriores.

4.1. Ciclo 1: Geração Baseada em Templates Estruturados

O primeiro ciclo iterativo foi estruturado a partir dos desafios documentados na literatura: a geração manual ou baseada em prompts genéricos resultava em questões com variação descontrolada de dificuldade, desalinhamento temático e alto custo de produção. Para resolver essas limitações, propôs-se a utilização de templates trifásicos, divididos em contexto, problema e variáveis. Essa abordagem visou, principalmente, controlar a fragmentação pedagógica, ou seja, garantir que as questões geradas fossem alinhadas a determinados tópicos e níveis de complexidade. Nesse ciclo foi utilizada a técnica de prompts encadeados (prompt chaining), decompondo uma tarefa complexa em subtarefas sequenciais e interligadas, onde a saída de um prompt serve como entrada para o próximo [Wu et al. 2022]. Primeiro houve a geração contextos amplos e abstratos (por exemplo, um cenário genérico para um tema específico), e posteriormente, para cada contexto criava-se problemas específicos e independentes. Um exemplo de prompt dividido em duas partes foi:

> **Geração do contexto:** Crie um **contexto amplo e abstrato** para uma questão sobre o tema **tema**. O contexto deve ser uma **descrição geral de uma situação real, sem definir problemas específicos.** Ele deve conter elementos que podem ser explorados para gerar múltiplos problemas.

Geração dos problemas:

> Gere um novo problema completamente diferente dos anteriores dentro deste contexto:
> {contexto}
> Instruções importantes:
> - O problema deve ser **completamente diferente** dos problemas anteriores gerados neste contexto.
> - Ele deve abordar **outro aspecto do contexto** e usar **variáveis diferentes**.

Esta abordagem resultou em questões que possuem contextos realistas e problemas bem definidos. A estrutura modular (contexto + problema) permite adaptação a outros temas, enquanto a simplicidade do enunciado garante acessibilidade para iniciantes, mantendo potencial para expansão em complexidade. Abaixo um exemplo de questão gerada neste ciclo:

> **TEMA:** Operadores e Operações Matemáticas
>
> **DIFICULDADE:** Fácil
>
> **CONTEXTO:** Uma empresa de tecnologia está desenvolvendo um sistema de gerenciamento de projetos para equipes de desenvolvimento de software. O sistema precisa lidar com várias funcionalidades, como a gestão de tarefas, alocação de

recursos, controle de prazos e geração de relatórios de progresso. As equipes são compostas por membros com diferentes funções, como desenvolvedores, designers e gerentes de projeto. Cada projeto possui um orçamento específico, um cronograma detalhado e um conjunto de objetivos a serem alcançados. A empresa deseja otimizar o processo de planejamento e execução dos projetos, garantindo que os recursos sejam utilizados de forma eficiente e que os prazos sejam cumpridos.

PROBLEMA: Uma equipe de desenvolvimento de software está trabalhando em um novo projeto que requer a colaboração de desenvolvedores, designers e gerentes de projeto. Cada membro da equipe contribui com um número específico de horas por semana. A empresa precisa calcular o total de horas semanais trabalhadas no projeto. Escreva um programa que receba o número de horas semanais trabalhadas por um desenvolvedor, um designer e um gerente de projeto. O programa deve calcular e exibir o total de horas trabalhadas pela equipe no projeto durante uma semana.

VARIÁVEIS: horas semanais do desenvolvedor, horas semanais do designer, horas semanais do gerente de projeto.

Apesar da estruturação via prompt chaining, uma avaliação um pouco mais detalhada de uma amostra das questões geradas identificou inconsistências significativas: nível de dificuldade discrepante (ex.: problemas introdutórios exigindo conceitos avançados como funções recursivas) e desalinhamento temático, como questões classificadas como ”condicionais” que demandavam uso de loops para resolução. Isso levou ao segundo ciclo de aprimoramento.

4.2. Ciclo 2: Introdução de Exemplos

Após a análise dos resultados do primeiro ciclo, identificou-se que, apesar da estruturação bem-sucedida com templates, as questões frequentemente apresentavam níveis de dificuldade inadequados, ora excedendo, ora ficando abaixo das expectativas pedagógicas definidas. Para enfrentar essa questão, decidiu-se empregar a técnica conhecida como *few-shot prompting* [Brown et al. 2020], que consiste em fornecer ao modelo exemplos concretos que possam orientar a geração, permitindo assim uma calibragem mais precisa da dificuldade.

Nesta abordagem, questões-modelo previamente validadas por especialistas foram incluídas explicitamente nos prompts utilizados pelo GPT-4. Tais exemplos serviram como referências diretas, ajudando o modelo a compreender e reproduzir padrões de complexidade desejados.

Um exemplo do prompt empregado nesta fase está apresentado a seguir:

Aqui está um exemplo de questão de programação introdutória sobre Operações Matemáticas e Entrada/Saída de Dados:
Exemplo:
Enunciado: Escreva um programa que receba dois números inteiros, calcule a soma deles e exiba o resultado.
Tarefa: Com base neste exemplo, gere uma nova questão sobre Operações

Matemáticas e Entrada/Saída de Dados com nível de dificuldade semelhante, utilizando variáveis diferentes e garantindo clareza no enunciado.

Como resultado, a introdução explícita desses exemplos de referência levou a uma melhoria significativa na adequação das questões ao nível esperado. Houve uma redução notável nos erros relacionados à complexidade excessiva ou insuficiente. Entretanto, a análise detalhada dos especialistas identificou novos desafios, principalmente relacionados à clareza, especificidade e completude dos enunciados, que ainda poderiam gerar ambiguidades para os alunos.

4.3. Ciclo 3: Refinamento Automático para Clareza e Completude

No terceiro ciclo, a preocupação central foi resolver os problemas remanescentes de clareza, especificidade e completude identificados anteriormente. Nesta fase, decidiu-se adotar uma técnica de engenharia de prompt conhecida como *self-refinement*, na qual o próprio modelo realiza uma revisão crítica automatizada das suas respostas iniciais [Madaan et al. 2023]. Este procedimento visa forçar o modelo a simular explicitamente o raciocínio de um programador humano, identificando e corrigindo ambiguidades e lacunas antes da finalização do enunciado.

A implementação consistiu em uma etapa adicional, onde o modelo avaliava sua própria saída inicial, considerando critérios objetivos previamente definidos pelos especialistas: clareza da linguagem, completude das instruções, especificidade da tarefa e definição clara das entradas e saídas. Para reforçar o aprendizado desse comportamento pelo modelo, foi novamente utilizada a técnica de *few-shot prompting*, agora com exemplos explícitos de questões antes e depois da correção.

Um exemplo do prompt de refinamento empregado neste ciclo é mostrado abaixo:

Analise rigorosamente a questão, considerando os critérios de clareza, completude e especificidade.

Questão Inicial:

Uma equipe está gerenciando um projeto de software. Os dados das tarefas estão numa lista. Implemente um programa que identifique as tarefas mais demoradas.

Problemas identificados: falta especificar claramente o formato dos dados de entrada (lista) e como serão fornecidos ao programa.

Questão Corrigida:

Uma equipe está gerenciando um projeto de software. Os dados das tarefas são fornecidos por meio de uma lista de inteiros digitados pelo usuário, cada número representando o tempo em horas gasto em cada tarefa. Implemente um programa que leia esses valores e exiba qual é a tarefa que demanda mais tempo.

Tarefa: Com base no exemplo acima, revise e melhore o seguinte enunciado [...]

O resultado deste terceiro ciclo foi notável em termos de qualidade textual das questões geradas. A aplicação do *self-refinement* praticamente eliminou ambiguidades relacionadas às instruções e especificações das questões. Os enunciados produzidos passaram a apresentar uma clareza significativamente maior, adequados para serem usados diretamente em contextos educacionais diminuindo consideravelmente a necessidade de ajustes adicionais pelos professores. A seguir temos um exemplo de questão refatorada de acordo com as instruções e exemplos do prompt:

Questão antes da correção:

Na metrópole do sudeste asiático, as autoridades estão desenvolvendo um sistema de transporte público integrando o BRT com monotrilhos. Suponha que você faz parte da equipe responsável por otimizar o uso de linhas de ônibus para atender às comunidades tradicionais sem comprometer seu patrimônio cultural. Os dados das linhas de ônibus são organizados em uma lista de listas, onde cada sublista contém três números representando, respectivamente, o número de viagens diárias, a capacidade média de passageiros e o número de paradas, por exemplo, [[20, 50, 10], [25, 60, 8], [15, 40, 12]]. Implemente um programa que percorra essa lista e identifique a linha de ônibus com o maior número de viagens diárias, a maior capacidade média de passageiros, e o maior número de paradas.

Problema encontrado:

Há uma ambiguidade na origem da lista (se é fixa, exemplo, ou entrada do usuário) e isso cria uma confusão estrutural na interpretação e na geração da solução.

Questão após a correção:

Na metrópole do sudeste asiático, as autoridades estão desenvolvendo um sistema de transporte público integrando ônibus BRT com monotrilhos. Você faz parte da equipe responsável por otimizar o uso das linhas de ônibus, garantindo o atendimento às comunidades tradicionais sem comprometer seu patrimônio cultural. Os dados das linhas de ônibus são fornecidos na forma de uma lista, onde cada elemento é uma sublista contendo exatamente três números inteiros positivos que representam, respectivamente: o número diário de viagens realizadas pela linha, a capacidade média de passageiros por viagem, o número total de paradas realizadas em cada viagem. Por exemplo: [[20, 50, 10], [25, 60, 8], [15, 40, 12]].

Adicionalmente, ao longo das discussões com a equipe durante o terceiro ciclo, concluiu-se que a definição de tema e dificuldade diretamente no prompt deixava margens para interpretação ambígua por parte do modelo. Isso ocorria especialmente pela subjetividade desses conceitos em contextos pedagógicos. A equipe passou então a considerar que esses elementos — tema e dificuldade — deveriam ser definidos a posteriori, a partir da análise do código gerado como solução para cada problema, uma vez que o conteúdo técnico efetivamente exigido fornece uma métrica mais objetiva e confiável.

Dessa forma, ao longo desses ciclos iterativos, a combinação e aplicação criteriosa das técnicas específicas de engenharia de prompt [Brown et al. 2020, Wu et al. 2022, Madaan et al. 2023] demonstraram ser altamente eficazes para a geração automatizada e robusta de questões pedagógicas para ensino introdutório de programação. Ao final dos

três ciclos, foram geradas e refinadas um total de 180 questões originais. A estratégia iterativa permitiu não apenas identificar e corrigir falhas em cada estágio, mas também consolidar um método eficaz de geração automática de questões com controle sobre complexidade, clareza e adequação pedagógica. A aplicação combinada de *prompt chaining*, *few-shot prompting* e *self-refinement* mostrou-se eficiente e replicável para contextos similares de produção de conteúdo educacional automatizado.

5. Lições aprendidas

A condução dos três ciclos de geração e refinamento de questões com auxílio de LLMs permitiu a identificação de um conjunto sólido de aprendizados sobre o uso de modelos generativos na formulação de questões de programação. Primeiramente, confirmou-se que a geração inicial de questões por IA, mesmo quando bem estruturada, raramente resulta em produtos prontos para uso imediato. O processo iterativo mostrou-se essencial, com cada ciclo revelando falhas que só puderam ser tratadas com refinamentos posteriores cuidadosamente conduzidos.

Um dos aprendizados mais valiosos foi a constatação de que prompts genéricos ou vagos comprometem seriamente a qualidade das saídas do modelo. Quanto mais detalhadas, explícitas e estruturadas forem as instruções no prompt — incluindo definição clara de entradas, saídas, variáveis, formatos de dados e até a presença de exemplos positivos e negativos — maiores são as chances de obter questões relevantes, claras e bem direcionadas. Nesse sentido, o uso de técnicas como *few-shot prompting* não apenas aprimorou a calibragem de dificuldade, como também reduziu a ambiguidade e a variabilidade indesejada.

Também se evidenciou que a estratégia de pós-processamento — em especial o *self-refinement* — é imprescindível quando o objetivo é alcançar um padrão elevado de completude e clareza. Esse processo, ao forçar o modelo a ”pensar como um programador”, ajudou a detectar e corrigir problemas que dificilmente seriam evitados apenas com um prompt inicial mais elaborado.

Outro aprendizado importante foi o abandono da tentativa de forçar o modelo a respeitar classificações de tema e dificuldade a priori. Como essas categorias são subjetivas e contextuais, optou-se por extrair essas informações a posteriori, com base no conteúdo da solução gerada. Essa inversão metodológica se mostrou mais robusta, já que está ancorada em dados objetivos (ex.: presença ou ausência de estruturas de repetição, tipos de operadores utilizados, etc).

Por fim, ficou claro que uma abordagem combinada — que envolve geração inicial, uso de exemplos bem escolhidos e mecanismos automáticos de avaliação e reescrita — supera amplamente abordagens isoladas. O uso de estruturas formais de avaliação (como a análise em formato JSON desenvolvida paralelamente) contribui para a automação e controle do processo, tornando-o mais escalável e confiável.

6. Conclusões

A resolução de exercícios, conforme ressaltado inicialmente, é um pilar fundamental no processo de aprendizado e retenção de conhecimento em computação, auxiliando na memorização, direcionando o foco do aluno, promovendo a recuperação de informações

e oferecendo *feedback*. Neste contexto, foi relatada aqui uma experiência com a geração automática de questões de programação através de LLM.

Foram realizados três ciclos de aperfeiçoamento no método de geração de questões, visando otimizar os resultados obtidos. Cada um desses ciclos promoveu melhorias significativas nos resultados, gerando diversas lições aprendidas que podem ser utilizadas por professores e pesquisadores de educação em computação. As 180 questões geradas durante esses ciclos foram incorporadas à base de questões de uma disciplina introdutória de programação do curso de graduação da UFRN.

Por fim, trabalhos futuros podem aperfeiçoar os prompts usados, por exemplo, desenvolvendo camadas adicionais de validação e considerando o feedback de alunos e professores em um processo interativo de geração. Mais especificamente, para a abordagem baseada em exemplos, pode ser avaliada a adoção de chamadas extras para a geração e avaliação das questões que servirão de exemplo, dispensando assim que as questões de exemplo sejam geradas por humanos.

References

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Carneiro, J., Aranha, E., and Santana, A. (2022). Aprendizado de domínio aplicada à educação matemática, da computação e engenharias: um mapeamento sistemático. In *XXXIII Simpósio Brasileiro de Informática na Educação*, Simpósio Brasileiro de Informática na Educação. Anais do XXXIII Simpósio Brasileiro de Informática na Educação.

Chan, W., An, A., and Davoudi, H. (2023). A case study on chatgpt question generation. In *2023 IEEE International Conference on Big Data (BigData)*, pages 1647–1656.

Doughty, J., Wan, Z., Bompelli, A., Qayum, J., Wang, T., Zhang, J., Zheng, Y., Doyle, A., Sridhar, P., Agarwal, A., Bogart, C., Keylor, E., Kultur, C., Savelka, J., and Sakr, M. (2024). A comparative study of ai-generated (gpt-4) and human-crafted mcqs in programming education. In *26th Australasian Computing Education Conference (ACE '24)*, pages 114–123, New York, NY, USA. Association for Computing Machinery.

Faraby, S. A., Romadhony, A., and Adiwijaya (2024). Analysis of llms for educational question classification and generation. *Computers and Education: Artificial Intelligence*, 7.

Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, and Sudha (2004). Design science in information systems research. *Management Information Systems Quarterly*, 28:75–.

Kurdi, G., Leo, J., Parsia, B., et al. (2020). A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*, 30:121–204.

Madaan, A., Lin, S., Liu, X., Yang, Y., Neubig, G., Le Bras, R., and Smith, N. A. (2023). Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Meißner, N., Speth, S., and Becker, S. (2024). Automated programming exercise generation in the era of large language models. In *36th International Conference on Software Engineering Education and Training (CSEET)*, pages 1–5, Würzburg, Germany.

Niu, Y. and Xue, H. (2023). Exercise generation and student cognitive ability research based on chatgpt and rasch model. *IEEE Access*, 11:116695–116705.

Thalheimer, W. (2003). The learning benefits of questions. Technical report, Work Learning Research. Tech. rep.

Wu, T., Jiang, E., Donsbach, A., Gray, J., Molina, A., Terry, M., and Cai, C. J. (2022). Promptchainer: Chaining large language model prompts through visual programming.