

Explorando *ChatGPT* para criação automática de questões práticas de Programação de Computadores

Fernando Euzebio Zimmerman¹, Fernando Henrique Oliveira Duarte¹,
Pedro Henrique Lopes Silva¹, Reinaldo Silva Fortes¹

¹ Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
35.402-136 – Ouro Preto – MG – Brasil

{fernando.zimmerman, fernando.hod}@aluno.ufop.edu.br
{silvap, reifortes}@ufop.edu.br

Abstract. *The increasing demand for coding skills in today's digital age has highlighted the importance of effective educational materials to help individuals learn how to code. Developing practical exercises for an introduction to programming can be challenging. Creating exercise datasets manually is a heavy task, but using Natural Language Processing (NLP) tools for text generation can automate this process and make it more scalable. This paper discusses the experiences in developing strategies to assess the feasibility of automatically creating introductory programming exercises by leveraging standard information and prompt engineering with NLP models.*

Resumo. *A crescente demanda por habilidades de programação na era digital atual tem destacado a importância de materiais educacionais eficazes que possam ajudar as pessoas a aprender a programar. Desenvolver exercícios práticos para uma introdução à programação pode ser desafiador. Criar conjuntos de exercícios manualmente é uma tarefa pesada, mas usar ferramentas de Processamento de Linguagem Natural (PLN) para geração de texto pode automatizar esse processo e torná-lo mais escalável. Este artigo discute as experiências no desenvolvimento de estratégias para avaliar a viabilidade de criar automaticamente exercícios introdutórios de programação com o ChatGPT, aproveitando a engenharia de informações e prompts padrão com modelos de PLN.*

1. Introdução

Nas disciplinas de programação de computadores dos anos iniciais do ensino superior, muitos alunos têm seu primeiro contato com a programação. A compreensão de conceitos abstratos, estruturas e sintaxes de uma linguagem de programação torna-se um desafio significativo no processo de aprendizagem do aluno [Gomes and Mendes 2007]. O desafio se estende para o professor que deve administrar o desenvolvimento de habilidades dos alunos no aprendizado de muitos algoritmos. O grande número de alunos por turma dificulta o acompanhamento individualizado por parte dos professores, afetando o aprendizado dos estudantes, resultando em baixo aproveitamento acadêmico e elevada taxa de evasão nessas disciplinas.

Sendo assim, deixar o aprendizado em programação eficiente e atrativo aumenta o desafio para os professores, demonstrando a necessidade dos educadores incorporarem tecnologias no auxílio do ensino, como demonstrado em [Thorell M. 2015]. Um dos

desafios é a criação de questões especializadas que se alinhem ao contexto específico do curso do aluno, ao mesmo tempo que abrangem uma diversidade de tópicos relevantes. Essas questões devem demonstrar como problemas da área de estudo do aluno podem ser resolvidos por meio da programação. Essa demanda evidencia a importância de estabelecer um banco de questões com uma ampla variedade de perguntas.

Cada atividade dentro do banco de questões e no contexto de disciplinas de programação possui um padrão bem definido. Os elementos de uma questão podem ser divididos em quatro: (1) tema a ser usado na questão; (2) dificuldade do tema; (3) recursos da programação a serem utilizados; e (4) dificuldade da implementação. Como há um padrão estabelecido, há formas de automatizar o processo de criação, sendo uma delas o uso de Processamento de Linguagem Natural (PLN).

PLN é uma área da Inteligência Artificial (IA) que explora a manipulação de textos em linguagem natural por computadores. Como mostrado em [Cambria and White 2014], a evolução das técnicas de PLN estão caminhando em direção a uma compreensão mais profunda e abrangente da linguagem humana por modelos computacionais. Desta forma, ferramentas implementadas com modelos de aprendizado de máquina no contexto de PLN, mais especificamente as que abrangem os *Large Language Models* (LLM), podem ser usadas para realizar a geração de uma variedade de novas questões de forma automática respeitando um padrão pré-definido.

A sinergia entre PLN e LLM é fundamental na geração automática de questões, especialmente no contexto educacional. Em [Kurdi et al. 2020], destaca-se que o uso de modelos de linguagem no processamento de textos possibilita a extração eficiente de informações relevantes para a formulação de questões. Por meio da aplicação de LLM, torna-se viável criar questões e usar essa integração para impulsionar o desenvolvimento de sistemas educacionais mais dinâmicos e personalizados, contribuindo significativamente para a melhoria da experiência de aprendizagem dos alunos e auxiliar os professores no ensino.

Recentemente, umas das arquiteturas de LLMs mais utilizadas é o *Generative Pre-trained Transformer* (GPT) através do *chatbot ChatGPT*, devido ao sucesso da ferramenta como mostrado por [Steve Mollman 2022] e sua exímia funcionalidade de conversação. O *chatbot* é uma aplicação que responde como uma entidade inteligente por meio de interações por texto ou voz, e compreende uma ou mais linguagens humanas. Os conteúdos passados para as ferramentas são chamados de *prompts* e podem ser melhorados para uma melhor eficácia em seu retorno através da engenharia de *prompt*, que é processo de criação e aprimoramento dessas instruções utilizadas pelo modelo. Através dela pode-se entender melhor os recursos e limitações dos LLMs, desenvolvendo e otimizando *prompts*, visando a eficiente utilização dos LLMs em diversas aplicações e áreas de pesquisa.

Sendo assim, a proposta deste trabalho é avaliar a utilização desta ferramenta, que utiliza o modelo generativo GPT, com o objetivo de contribuir com a geração automática de questões de programação, considerando os diferentes contextos das áreas da graduação, com a intenção de criar um banco de questões para complementar disciplinas introdutórias de Programação de Computadores. Este trabalho demonstra que o uso de LLM, como o GPT, pode auxiliar na geração automatizada de questões de programação. Embora seja necessária revisão humana para aprimorar a diversidade e qualidade das questões,

como mostrado em [Speth et al. 2023], os *chatbots* baseados em LLM podem ser eficazes na criação de exercícios práticos de programação. Além disso, este estudo ressalta a importância de explorar e adaptar ferramentas existentes, como o *ChatGPT*, por meio de interfaces personalizadas e APIs disponíveis para melhorar a diversidade e a qualidade das questões geradas para atender às diferentes demandas dos estudantes.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta trabalhos que inspiraram o atual trabalho. A Seção 3 apresenta a metodologia utilizada. A Seção 3.1 apresenta as estratégias usadas nos experimentos realizados. A Seção 4 apresenta os testes realizados e os resultados obtidos e, por último, a Seção 5 apresenta conclusões e propostas de trabalhos futuros.

2. Trabalhos Relacionados

Aqui destacamos trabalhos que inspiraram este estudo. Apesar da recente expansão no uso das LLMs, já é notável o grande interesse em explorar a capacidade da automatização em relação à educação.

Em [Sarsa et al. 2022], os autores usaram o *OpenAI Codex* [OpenAI 2024b], modelo semelhante ao *GPT-3* aprimorado com códigos disponíveis no *Github* focado na criação de código-fonte, para a criação automática de códigos e explicações de códigos. O estudo avaliou a eficácia desse modelo na geração de códigos como conteúdo educacional, com potencial de aplicação direta em cursos de programação. O trabalho demonstra que, ao fornecer palavras-chave específicas, é possível influenciar os conceitos de programação e temas contextuais abordados pelos códigos gerados.

Em [Finnie-Ansley et al. 2022], os autores avaliam o desempenho do *OpenAI Codex* em questões reais de exames de programação introdutória, comparando seus resultados com os de estudantes humanos. Eles também exploram como o *OpenAI Codex* lida com variações sutis na formulação dos problemas, utilizando versões publicadas do conhecido “*Rainfall Problem*”. Os resultados mostram que o modelo não apenas supera a maioria dos estudantes em exames, mas também demonstra uma significativa variação nas soluções geradas, adaptando-se bem a diferentes contextos e níveis de detalhe.

Os dois trabalhos anteriores apresentam a viabilidade no uso de LLMs para geração automática envolvendo códigos de programação. A principal diferença deste trabalho é que, nos trabalhos apresentados, os mesmos tiveram seu foco na geração do código. Neste trabalho, o objetivo é analisar a construção do texto que forma o enunciado do exercício de programação a ser resolvido.

Em [Speth et al. 2023], explora-se o uso do *ChatGPT* para gerar exercícios de programação voltados para estudantes iniciantes e intermediários, com ênfase em conceitos de programação orientada a objetos. Os autores adotaram um processo iterativo, começando com *prompts* iniciais que forneciam o contexto do curso e os objetivos de aprendizagem desejados. Caso o resultado não atendesse às expectativas, os *prompts* eram ajustados para orientar melhor a geração de conteúdo, e uma nova iteração era realizada até que o exercício gerado estivesse conforme o esperado. O estudo também ressalta a importância de fornecer contexto adequado nas solicitações para otimizar a qualidade das saídas geradas pela ferramenta.

Assim como em [Speth et al. 2023], buscamos realizar interações com a ferramenta

usando o contexto da disciplina para obter resultados guiados à nossa estrutura de questão objetivo, refinando as entradas e chegando a um modelo de *prompt* mais direto. Dessa forma, o nosso objetivo é também enriquecer os enunciados das questões, oferecendo uma introdução clara ao assunto e aos tópicos de programação abordados visando proporcionar uma melhor compreensão e experiência para o aluno, facilitando a resolução das questões. Além disso, nosso foco está em programação introdutória, não restringindo o escopo a tópicos específicos, como o de orientação a objetos.

3. Metodologia

Para o desenvolvimento da metodologia proposta, utilizou-se como base o conteúdo estudado na disciplina de Programação de Computadores ministrada na Universidade Federal de Ouro Preto. Utilizou-se como referência o livro texto da disciplina¹, além de um pequeno conjunto de questões já usadas. O objetivo é incorporar elementos que enriqueçam as questões e contribuam na eficácia da geração das ferramentas.

Para a geração das questões, o seguinte padrão foi adotado: (1) **Cabeçalho**: composto por um título e subtítulo que caracterizem a questão; (2) **Tema**: tema geral da questão apresentada; (3) **Recursos necessários**: indicar quais tópicos de programação abordados na disciplina estão relacionados à questão e os recursos da linguagem que serão necessários para a solução da questão; (4) **Nível de dificuldade**: apresentar o nível que se considera a questão, conforme a escala definida. A métrica do nível de dificuldade varia em uma escala de 1 a 10, onde 1 é o nível mais fácil e 10 o nível mais difícil. A identificação da dificuldade ficou a critério do próprio modelo e, posteriormente, se necessário, fica a critério do usuário aceitá-la ou ajustá-la; (5) **Introdução**: parte introdutória sobre a questão, visando contextualizar a questão; (6) **Especificações**: informações gerais necessárias e condições a serem consideradas para a solucionar a questão; (7) **Exemplos de execução**: exemplos do que seriam as saídas ou o funcionamento geral esperado do programa; (8) **Solução**: uma possível solução para a questão. A solução não fará parte do enunciado disponibilizado para os alunos e servirá apenas como referência para quem está elaborando as questões. A Figura 1 mostra um exemplo do que seria uma questão completa com a identificação de cada componente.

Após a definição do padrão de questão, estabeleceu-se o arcabouço metodológico deste estudo, delineando-se os procedimentos experimentais a serem adotados. A metodologia consiste em experimentos feitos mediante pedidos passados via *chat* por meio da interface do próprio *ChatGPT* e, conforme a resposta gerada, realiza-se uma análise comparativa considerando o padrão de questão adotado. A partir daí, os pedidos eram adaptados a fim de receber sugestões de questões cada vez mais próximas ao padrão estabelecido.

Para os experimentos, o *ChatGPT* na versão 3.5 [OpenAI 2024a] é utilizado como ferramenta para a geração de questões. O *ChatGPT* é uma ferramenta que possui o formato de *chatbot*, um algoritmo que simula uma conversa com o usuário, que usa o modelo GPT para gerar respostas para perguntas, solicitações e comentários. Os testes foram conduzidos sequencialmente e a estratégia de interação adaptativa, sendo utilizada para desenvolver os *prompts* associados a outra estratégia.

¹Disponível em <http://www.decom.ufop.br/bcc701/material/>.

(2) Física (MRU)

(5) A função do movimento uniforme nos permite encontrar a posição (S) de um objeto que se movimenta com velocidade constante (v) em qualquer instante de tempo (t), desde que saibamos sua posição inicial (Si), através da seguinte fórmula:

$$S = S_i + v \cdot t$$

(1) Questão 1.1 - (3) Expressões e IO - (4) 1

(6) Implemente um programa que solicite os valores de entrada aos usuários e os armazene nas variáveis: posição inicial (Si), a velocidade (v) e o instante de tempo (t), respectivamente. Todas as entradas são números reais. O programa deve realizar o cálculo da posição final (S) e imprimir o resultado no terminal com uma precisão de 2 casas decimais.

Sua resposta deve conter as mensagens exatamente como definidas nos exemplos de execução.

(7) Exemplo de Execução 1

```
Informe a posição inicial (Si): 10
Informe a velocidade (v): 1
Informe o instante de tempo (t): 5
A posição final no tempo t = 5.00 será S = 15.00
```

Exemplo de Execução 2

```
Informe a posição inicial (Si): 55.7
Informe a velocidade (v): 60
Informe o instante de tempo (t): 5.8
A posição final no tempo t = 5.80 será S = 403.70
```

(8) Solução:

```
1. Si = float(input('Informe a posição inicial (Si): '))
2. v = float(input('Informe a velocidade (v): '))
3. t = float(input('Informe o instante de tempo (t): '))
4. S = Si + v * t
5. print(f'A posição final no tempo t = {t:.2f} será S = {S:.2f}')
```

Figura 1. Exemplo de questão usada.

3.1. Estratégias para a geração de questões

Nesta seção, é discorrido sobre os objetivos das estratégias de testes empregadas neste trabalho. O fluxograma apresentado na Figura 2 representa o fluxo dos testes.



Figura 2. Fluxograma de representação dos experimentos.

Todos os experimentos seguiram um fluxo de experimentação uniforme, com a única distinção sendo as particularidades específicas de cada estratégia aplicada antes

da entrada dos dados nas ferramentas. Onde, interage-se com a ferramenta por meio de solicitações específicas, analisando os retornos obtidos. Caso necessário, os pedidos são adaptados para otimizar os resultados, ou uma nova interação é iniciada com ajustes nas instruções, ou contexto, visando aprimorar a qualidade das questões geradas. A avaliação de retorno satisfatório foi feita sob a perspectiva dos autores deste trabalho.

Foram exploradas duas estratégias de interação com o modelo: (1) Interação adaptativa; e (2) Estrutura de *prompt*. Cada uma visa aprimorar a eficácia das respostas fornecidas pelo modelo, seja adaptando os pedidos de forma dinâmica, refinando as instruções passadas ao modelo, ou utilizando exemplos para contextualização. Estas estratégias são descritas a seguir:

- **Interação adaptativa:** A estratégia de Interação Adaptativa, executada na ferramenta *ChatGPT*, tornou-se a base para as demais estratégias, uma vez que foram exploradas a maioria das possibilidades de retorno do modelo nesta ferramenta. A Interação adaptativa consiste em interagir com a ferramenta realizando um pedido e, a partir do retorno obtido, realizava-se uma análise onde, se necessário, o pedido atual era adaptado para buscar um melhor retorno, caso contrário, partia-se para uma nova interação com um novo pedido.
- **Estrutura de *prompt*:** A estratégia de Estrutura de *prompt*, executada na ferramenta *ChatGPT*, se baseia em conceitos de Engenharia de *prompt*, sendo o processo de criação e aprimoramento dos *prompts*, que representa uma instrução passada em linguagem natural que orienta um LLM durante a execução de uma tarefa específica, na intenção de melhorar a interação com o modelo e obter uma eficácia melhor nas respostas. A estrutura era composta por um contexto, instrução e modelo de resposta estáticos, onde eram mudados os campos referentes aos recursos considerados na criação da questão. A resposta era analisada e em seguida novas informações eram atribuídas aos campos dinâmicos e o pedido era feito novamente.

4. Testes e Resultados

Nesta seção são apresentadas informações a respeito dos testes realizados com as ferramentas e os resultados obtidos por meio das estratégias aplicadas na Subseção 4.1. Já a Subseção 4.2, apresenta reflexões realizadas a partir dos resultados dos testes.

4.1. Testes realizados

As subseções a seguir, relatam os testes para cada estratégia: interação adaptativa na Subseção 4.1.1; e estrutura de *prompt* na Subseção 4.1.2.

4.1.1. Interação adaptativa

Iniciando a exploração do potencial do *ChatGPT* para criar questões de programação, um experimento foi conduzido, solicitando ao modelo a geração de questões com base em detalhes básicos fornecidos. Os retornos obtidos foram utilizados para ajustar os pedidos, feitos via *chat*, a fim de alinhar-se ao padrão esperado de questão. Esses ajustes incluíram a adição de contexto, exemplos adicionais e uma solução, com destaque para mais exemplos de execução para clarificar a resposta esperada.

Em paralelo, foi implementada a estratégia de aquecimento do modelo de linguagem, que envolve a apresentação de exemplos prévios para alinhar o modelo ao padrão

desejado de resposta. Desta forma, usou-se como exemplo a questão apresentada na Figura 1. Mesmo após as interações realizadas e passando a questão com os dados desejados, a ferramenta não havia atingido exatamente o padrão de saída do exemplo usado como aquecimento, como mostrado no trecho a seguir:

ChatGPT: Com base na estrutura da questão fornecida, aqui estão mais três questões criadas seguindo o mesmo formato:

Questão 1: Implemente um programa em Python que calcule a posição final de um objeto em um movimento uniforme. O programa deve solicitar ao usuário a posição inicial (S_i), a velocidade (v) e o instante de tempo (t). Verifique se todas as entradas são números reais maiores que zero. Caso sejam válidas, realize o cálculo da posição final (S) e imprima o resultado no terminal com uma precisão de 2 casas decimais. Caso contrário, exiba a mensagem "Entradas inválidas".

Exemplo de Execução:

Informe a posição inicial (S_i): 20
Informe a velocidade (v): 2.5
Informe o instante de tempo (t): 8.75
A posição final no tempo $t = 8.75$ será $S = 41.25$

...

Seguindo com os testes explorativos, em um deles, pode-se destacar um ponto interessante com relação ao retorno da ferramenta. O pedido tinha como objetivo a geração de uma questão com o tema MRU (Movimento Retilíneo Uniforme) que envolvesse o tópico de repetição *while*. Obteve-se uma confirmação de que a ferramenta geraria uma questão com este tópico, algo que não ocorreu, como mostra o trecho a seguir:

ChatGPT: [...] Nesta questão, o programa solicita ao usuário a distância a ser percorrida e a velocidade do objeto. Em seguida, utiliza a fórmula do MRU para calcular o tempo necessário para percorrer a distância informada. O resultado é exibido no terminal. Note que não foi necessário utilizar a estrutura de repetição *while* nesta questão específica. Caso você deseje uma questão com *while*, por favor, me informe para que eu possa ajustar a solução.

Usuário: Eu quero uma questão com *while*

ChatGPT: Desculpe pelo equívoco anterior. Aqui está uma questão sobre MRU que utiliza a estrutura de repetição *while*:

...

Buscando entender melhor o funcionamento da ferramenta, usou-se dela própria, questionando como ela gerava suas respostas e solicitando o padrão que ela mesma usava para criar as questões. O *ChatGPT* retornou as diretrizes consideradas na criação das questões, foram elas: (1) **Compreensão do tema:** tentativa de entender o tema proposto pelo usuário para criar questões relevantes dentro desse contexto; (2) **Estrutura e formato:** tentativa de criar questões com uma estrutura clara e coerente; (3) **Variedade e desafio:** tentativa de criar questões variadas em termos de dificuldade, tópicos e requisitos; (4) **Personalização:** tentativa de incorporar as preferências e requisitos do usuário sempre que possível; (5) **Validade e precisão:** esforço para garantir que as questões geradas sejam válidas e corretas em relação ao tema proposto.

Posteriormente, buscou-se adicionar diretrizes para adaptar a geração de questões, foram elas: (1) **Dificuldade da questão:** diretriz responsável para indicar a complexidade

do algoritmo a ser desenvolvido com base nos recursos da linguagem usados e também o tema da questão; (2) **Fonte usada na criação da questão**: diretriz responsável para indicar de onde foi a base e de onde foram pegados os recursos para a geração da questão; (3) **Análise de plágio**: diretriz responsável em ser um modo de evitar qualquer possibilidade da questão gerada ser completamente plagiada, evitando problema de *copyright*.

Em relação às diretrizes adicionadas, pediu-se para que a ferramenta nos informasse cada uma delas nas questões geradas. Foi analisada a própria consideração da ferramenta na aplicação dessas diretrizes. Dessa forma, a respeito da diretriz de dificuldade, foi difícil analisar o processamento e motivações considerados pela ferramenta ao definir uma dificuldade, e foi possível notar que a consideração estava mais ligada ao tema do que aos recursos da linguagem usados

E, em relação à diretriz da fonte, as respostas eram bem genéricas como, por exemplo, se tratando de questões com MRU, a fonte retornada era simplesmente “Material didático de Física”. Especificamente na diretriz relacionada ao plágio, a ferramenta informou que não possuía essa capacidade.

Após as análises e adaptações com relação às diretrizes, seguiu-se um tempo sem interação com a ferramenta antes de prosseguir com novos testes, o que resultou em retornos sem a consideração de todo o contexto previamente informado. Constatando que a mesma não possuía memória de contexto. Para concluir, a própria ferramenta informou que não possuía essa capacidade, como mostra o trecho a seguir:

Usuário: Você considera toda a nossa conversa em futuras conversas?

ChatGPT: Como modelo de linguagem, não tenho a capacidade de lembrar ou armazenar informações sobre conversas passadas. Cada interação comigo é independente e não tenho a capacidade de recordar ou utilizar informações de conversas anteriores. Cada vez que você inicia uma nova conversa, é como se estivéssemos começando do zero, sem memória das interações anteriores. Essa abordagem foi projetada para garantir a privacidade e segurança dos usuários. Portanto, é importante sempre fornecer contexto e informações relevantes em cada interação para que eu possa responder da melhor maneira possível.

Resultados: Os resultados referentes à estratégia de Interação adaptativa se mostraram positivos ao considerar o padrão de questão que se desejava alcançar, apesar de alguns pontos a serem considerados. Quanto mais interações eram feitas, mais as questões eram retornadas no formato esperado e com certeza serviriam de auxílio para um docente em uma criação de questões. Entretanto, interagir como modo de conversa com o modelo e ir ajustando as questões conforme se espera por meio de um “bate-papo”, pode não ser eficiente, além de que, como mostrado durante os resultados, nem sempre era fiel ao pedido feito como quando solicitado a questão que usasse o recurso *while* e o mesmo não retornou, sendo necessário mais interações para atender o pedido. Outro ponto é que, durante os testes, nem sempre ele retornava a quantidade de questões esperadas, como quando era pedido questões que usassem mais do que um tópico e se esperava que o mesmo retornasse uma para cada, só que, por vezes isso não ocorria, sendo necessário mais interações pedindo para que a ferramenta continuasse a gerar. Ainda, podemos citar uma das principais causas que tornam essa estratégia não muito eficiente, a não retenção de memória do modelo, o que obriga ao usuário repetir todas as informações e ir alcançando o aperfeiçoamento da questão em todas as vezes que desejar criar uma nova questão, seja em novo *chat* ou no mesmo após decorrido longo tempo sem interação.

4.1.2. Estrutura de *prompt*

Após os testes de forma de conversação com a ferramenta, visando uma melhor compreensão do que esperávamos por meio da mesma, a busca de melhores respostas e uma padronização nos pedidos adotou-se um modelo seguindo uma abordagem voltada à engenharia de *prompt*. A estratégia foi usada com objetivo de resolver o problema de sempre realizar várias interações com a ferramenta para chegar na questão desejada. Desta forma, a geração das questões poderia ser mais direta realizando apenas uma única interação, personalizável onde havia a necessidade de mudar apenas pontos específicos na estrutura do *prompt*, como o tema da questão e os recursos a serem considerados, e escalável com a inserção de vários tópicos de uma vez ou inserindo vários novos *prompts* de forma rápida., considerando os seguintes elementos: (1) **Contexto**: no qual é passado a informação para a ferramenta que envolva informações gerais considerando o contexto em questão para direcioná-lo a melhores respostas; (2) **Instrução**: uma tarefa ou instrução específica que você deseja que o modelo execute; (3) **Dados de entrada**: entrada para a qual se espera encontrar uma resposta; (4) **Indicador de saída**: onde é indicado o formato em que ele responda.

A partir da estrutura desses elementos e do objetivo final do trabalho, foi adotado o modelo de *prompt* representado pelo Quadro:

Contexto
Contexto: Assuma o papel de um professor da matéria de programação de computadores com Python para turmas de cursos de graduação em engenharia. Matéria: Física. Assuntos da matéria: MRU, Termodinâmica. Tópico de programação: Variáveis, expressões, entradas e saídas; Decisão simples (if, else); Decisão complexa (if, elif, else); Repetição e Laços; Laços aninhados; Funções; Estrutura de dados homogêneos com vetores; Estrutura de dados homogêneos com matrizes; Estrutura de dados heterogêneos (registros).
Instrução
Instrução: Sempre crie uma questão de programação para cada <Tópicos de programação> utilizando os <Assuntos da matéria> da disciplina de <Matéria>. Garanta que cada questão aborde um assunto específico e que todos os tópicos de programação listados sejam contemplados. Classifique as questões usando um medidor de dificuldade de 1 a 10, onde 1 é a questão mais fácil e 10 a mais complexa.
Modelo de resposta
Use o seguinte formato de resposta: <matéria> - <assunto da matéria abordado> Questão <número da questão> - <tópico usado> - <dificuldade da questão> Contextualização e introdução sucinta do assunto da matéria abordado, tal qual explicações prévias dos recursos utilizados no desenvolvimento da questão. Enunciado da Questão. Exemplos: Três exemplos de execução e da saída que se espera do programa. Solução:

Uma solução possível.

A partir daí, foram realizados testes com intuito de obter resultados mais diretos, onde somente bastava modificar a matéria, assuntos e tópicos de programação abordados no pedido e enviar o *prompt* ao *ChatGPT*, como apresentado:

Usuário: [inserção do prompt onde: matéria = Física; Assuntos da matéria = MRU, Termodinâmica; Tópicos = Variáveis, expressões, entradas e saídas, Decisão simples (if, else), Decisão complexa (if, elif, else), Repetição e Laços, Laços aninhados, Funções, Estrutura de dados homogêneos com vetores, Estrutura de dados homogêneos com matrizes, Estrutura de dados heterogêneos (registros)]

ChatGPT: Física - MRU

Questão 1 - Variáveis, Expressões, Entradas e Saídas - Dificuldade 2

Contextualização: No estudo do Movimento Retilíneo Uniforme (MRU), é comum utilizar equações para calcular a posição de um objeto em função do tempo. Vamos criar um programa em Python para calcular e exibir a posição de um objeto em MRU.

Enunciado da Questão: Dado o MRU de um objeto, onde a posição inicial é dada por $posicao_{inicial}$, a velocidade é dada por velocidade e o tempo é dado por tempo, escreva um programa em Python que calcule e imprima a posição do objeto.

Exemplos:

$posicao_{inicial} = 10$

velocidade = 5

tempo = 2

Saída esperada: A posição do objeto após 2 segundos é 20 metros.

...

Resultados: Os resultados referentes à estratégia de Estrutura de *prompt* se mostram positivos e se assemelham aos alcançados pela estratégia Interação adaptativa. Entretanto, ter uma entrada padrão de pedido torna o processo mais eficiente.

Os resultados obtidos ao alterar os campos em que consideramos dinâmicos foram consideravelmente positivos. Porém, nem sempre, o modelo gerava a quantidade de questões esperadas. Isso pode acontecer pela instrução não definir uma quantidade exata e por esperar que o modelo considerasse a quantidade de tópicos e fizesse a ligação com a instrução de criar questões para todas as quantidades.

Ainda assim, com um *prompt* bem elaborado e instruções mais específicas, deve ser possível alcançar resultados melhores e mais próximos ao que uma questão final almeja.

4.2. Reflexões obtidas

Apesar de, no geral, os resultados serem positivos, notam-se alguns pontos que poderiam ser melhorados e que precisam de ajuste manual no desenvolvimento das questões, descritos a seguir:

- **Melhorar introdução e Contextualizações:** embora as contextualizações sejam adequadas, por vezes elas poderiam ser mais ricas e detalhadas, proporcionando aos alunos uma experiência mais ampla na aplicação dos conceitos abordados.
- **Criação completa das questões:** embora o *prompt* especifique uma lista de tópicos de programação e o pedido peça para criar para cada um deles, a maioria das respostas não contempla todos os tópicos. Nota-se, entretanto, que quando o número é fixo para um objetivo mais específico, como a criação de uma prova contendo 5 questões, as ferramentas tendem a criar todas.

- **Redundância nas Questões:** por vezes, as questões geradas acabam sendo bastante semelhantes, abordando os mesmos tópicos e temas.
- **Diversificação nos Exemplos:** os exemplos fornecidos nas questões possuem bastante similaridade, indicando uma falta de diversidade de exemplos que envolvam as diferentes situações apresentadas.
- **Abordagem de Casos Especiais:** a inclusão de casos especiais nas questões, como situações limite ou cenários não triviais, poderia desafiar os alunos a pensar de maneira mais crítica e a aplicar os conceitos de programação de forma mais abrangente.
- **Exemplos de Código mais Complexos:** embora os exemplos de código fornecidos sejam claros, poderiam ser adicionados casos mais complexos que gerassem uma discussão de implementações alternativas, podendo contribuir para questões mais enriquecedoras.
- **Incorporação de Boas Práticas:** o modelo poderia explicitamente destacar boas práticas de programação, como comentários descritivos, modularização do código, e considerações sobre eficiência e legibilidade, para promover uma abordagem mais consistente quanto ao desenvolvimento de software.

5. Conclusão e Trabalhos futuros

O estudo conclui que o modelo GPT, implementado na ferramenta testada, é viável para auxiliar na criação de questões automatizadas. Apesar de revelar fragilidades em relação à abrangência de temas, tópicos e dificuldade das questões, necessitando de revisão humana para melhorar os resultados, o modelo cumpre o papel de auxiliar a criação de questões de programação.

Entretanto, observou-se dificuldade do modelo em gerar todas as questões de forma completa e consistente, além de uma tendência redundante em suas saídas, o que pode prejudicar a diversificação dos exemplos e impactar negativamente na qualidade do banco de questões, comprometendo a riqueza e variedade necessárias para atender às diferentes demandas dos estudantes. Também houve limitação na abordagem de casos especiais da programação por parte do modelo GPT, que apresentou falta de profundidade nas soluções propostas, comprometendo a utilidade prática das questões geradas em ambientes educacionais. A falta de acesso direto à estrutura das ferramentas e às diretrizes do modelo impede o ajuste preciso das saídas, afetando a garantia da diversidade das questões e dificultando a automação do processo.

A partir dos experimentos e resultados obtidos por meio deste trabalho, têm-se algumas sugestões para a continuidade do mesmo. Como: **(1) Interface Própria:** que receba um *prompt* adaptado a partir do *prompt* definido inicialmente como base; **(2) Usar APIs:** algumas ferramentas como o *ChatGPT*, possuem APIs disponíveis, possibilitando uma maior exploração do modelo na intenção de melhorar o seu desempenho; **(3) Explorar outros modelos:** seguindo essa linha de pesquisa, podem ser exploradas inteligências artificiais que incorporem outros modelos de linguagem, como o *Gemini* e o *Maritalk*; **(4) Aplicar *fine-tuning*:** aprimorar os resultados a partir da especialização de modelos existentes; **(5) Ampliar experimentos:** aprofundar as análises experimentais com a participação de voluntários diversificados, como professores e alunos das disciplinas de programação; **(6) Sistema com banco de dados:** realizar a implementação de um sistema de criação automática de questões que possua um banco de dados onde as questões sejam

armazenadas, na intenção de criar um repositório robusto e organizado para facilitar o acesso futuro e a reutilização do conteúdo.

Agradecimentos

Os autores gostariam de agradecer à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Universidade Federal de Ouro Preto (PROPPI/UFOP) pelo apoio ao desenvolvimento deste estudo.

Referências

- Cambria, E. and White, B. (2014). Jumping nlp curves: A review of natural language processing research [review article]. IEEE Computational Intelligence Magazine, 9(2):48–57.
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In Proceedings of the 24th Australasian Computing Education Conference, ACE '22, page 10–19, New York, NY, USA. Association for Computing Machinery.
- Gomes, A. and Mendes, A. (2007). Learning to program - difficulties and solutions. In International Conference on Engineering Education–ICEE, pages 283–287.
- Kurdi, G., Leo, J., Parsia, B., Sattler, U., and Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. International Journal of Artificial Intelligence in Education, 30:121–204.
- OpenAI (2024a). ChatGPT. Disponível em: <https://openai.com/chatgpt>. Acessado em Abril de 2024.
- OpenAI (2024b). Openai codex. Disponível em: <https://openai.com/index/openai-codex>. Acessando em Abril de 2024.
- Sarsa, S., Denny, P., Hellas, A., and Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using large language models. In Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1, ICER '22, page 27–43, New York, NY, USA. Association for Computing Machinery.
- Speth, S., Meißner, N., and Becker, S. (2023). Investigating the use of ai-generated exercises for beginner and intermediate programming courses: A chatgpt case study. In 2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T), pages 142–146.
- Steve Mollman (2022). Chatgpt gained 1 million users in under a week. here's why the ai chatbot is primed to disrupt search as we know it. Disponível em: <https://finance.yahoo.com/news/chatgpt-gained-1-million-followers-224523258.html>. Acesso em: 27 de junho 2023.
- Thorell M., Fridorff-Jens P.K., L. P. e. a. (2015). Transforming students into digital academics: a challenge at both the individual and the institutional level. BMC Med Educ 15, 48 (2015). <https://doi.org/10.1186/s12909-015-0330-5>.