Construção de um Modelo de Desenvolvimento/Geração de Código com IA

Gustavo Orlando Costa dos Santos Henriques - 64361
Estudo Orientado
Mestrado em Engenharia Informática
Faculdade de Ciências, Universidade de Lisboa
fc64361@fc.ul.pt

Resumo

Palavras-chave *Inteligência artificial; Grandes modelos de linguagem (LLMs); Engenharia de prompts; Geração automática de código; Engenharia de software*

1 Introdução

Outline. How is the rest of the document structured?
The remainder of this document is organised as follows. Section 2 presents bla bla bla. ...

2 Enquadramento Teórico

Esta secção serve como base teórica para clarificar alguns conceitos importantes, oferecendo uma melhor compreensão das secções que se seguem.

2.1 Geração automática de código

2.2 Engenharia de prompts

citação e descrição por palavras tuas

Para entedermos este conceito é primeiro necessário perceber a definição de prompt. Segundo Marvin et al.[5] "A prompt is a text-based input that is fed to a language model to guide its output. A prompt can be audio but, in this case, the audio input would be transcribed into text and fed to the language model as a text-based prompt. The language model would then process the text-based prompt and generate an output based on the instructions and context provided in the prompt...", ou seja, um prompt é um input em formato textual (ou transcrição de outro meio, como áudio ou imagens) usado para orientar a saída de um modelo artificialmente inteligente, servindo para fornecer instruções e contexto, de modo a que o mesmo gere uma resposta em conformidade com a tarefa desejada.

Uma das influências na qualidade de resposta é a forma como são construídos estes prompts, visto que pequenas alterações nos mesmos são capazes de gerar respostas bastantes diferentes[6]. Assim, por volta de 2022, emergiu esta nova disciplina no campo da inteligência artificial, que tem como objetivo conceber prompts e otimiza-los, tornando o uso de LLMs mais eficiente[5]. Outra das razões para ter surgido o prompt engineering deve-se ao facto do custo de treinamento dos modelos. À medida que a complexidade dos modelos aumenta, os gastos em treinamento tornam-se cada vez maiores. Um estudo afirma que o custo associado aos modelos mais computacionalmente intensivo, tem crescido a uma taxa de 2.4 vezes ao ano desde 2016[2]. ~~No entanto, ao~~ usar estas técnicas, conseguimos afinar os LLMs sem gastar recursos para treinamento. Com isto, é notável a importância das mesmas visto que conseguem melhorar a qualidade do output sem necessitar de um dispendioso processo de treinamento.

chamada prompt engineerig

treino

quais?

nao acrescenta nada de novo ao que disseste antes

Desde 2022 têm vindo a ser desenvolvidas várias estratégias de prompt engineering. Estas estratégias têm sempre um propósito específico que querem melhorar, desde a melhoria do raciocínio e lógica do modelo, até à redução de alucinações ou à geração e execução de código[8]. Dependendo de cada estratégia utilizada, o prompt é construído de maneiras diferentes. No entanto, existem boas práticas gerais para a elaboração de prompts efetivos. De acordo com Marvin et al.[5], definir o objetivo, compreender as capacidades do modelo e fornecer contexto são alguns dos passos envolvidos na criação de um prompt eficaz.

exemplos?

3 Trabalho relacionado

A presente secção reúne o estado da arte relevante para esta tese, apresentando trabalhos, modelos e ferramentas que abordam problemas relacionados com a aplicação de métodos de inteligência artificial ao desenvolvimento de software, a utilização de técnicas de prompting e avaliação de modelos.

3.1 Geração Automática de Código

Nesta secção são analisados trabalhos que ilustram várias abordagens para automatizar a transformação de descrições diretamente em código.

Xu et al.[10] propõem um sistema que converte imagens de interfaces web em código HTML e CSS. O método assenta numa pipeline composta por três etapas principais: geração de um dataset com imagens e respetivo código, deteção dos componentes visuais da interface recorrendo a modelos como CNN[1] e Faster R-CNN[2] e, produção do código através de uma

[1] https://cs231n.github.io/convolutional-networks/
[2] https://medium.com/@RobuRishabh/understanding-and-implementing-faster-r-cnn-248f7b25ff96

melhor começar pelo mais geral; isto ja é uma coisa que atua num nicho, deve vir já depois das mais gerais

arquitetura que combina uma CNN para extração visual com uma LSTM[3] responsável por gerar a sequência de código. Este trabalho é relevante para a presente dissertação por demonstrar uma abordagem para a transformação do design em estruturas front-end executáveis, alinhando-se com a componente da pipeline dedicada à geração automática da interface.

Hoje em dia já existem vários modelos generativos com a finalidade de desenvolvimento de código, ou seja, modelos que foram construídos com o objetivo principal de gerar como resposta um excerto de código que cumpre os requisitos solicitados pelo ser humano. Codex[1] foi um dos primeiros modelos amplamente conhecidos para transformar descrições textuais em código executável, suportando múltiplas linguagens e tarefas como geração de funções, completamento e tradução entre linguagens. Este modelo serviu ainda de base ao GitHub Copilot, que será abordado mais adiante, estabelecendo um marco na utilização prática de LLMs para auxiliar o desenvolvimento de software.

completação ou completamento? Escolhe uma e usa sempre a mesma

Dando seguimento a esta ideia de ter modelos especializados na geração de código, surgiu também o CodeLlama[7], uma família de modelos orientados para tarefas de programação como geração, explicação e preenchimento de código. Estes modelos, que foram desenvolvidos com base no modelo Llama2[9], são disponibilizados em três variantes distintas:

Code Llama, a versão base, implementada para tarefas mais gerais de compreensão e transformação de código;

Code Llama - Python, uma versão treinada adicionalmente com grandes bases de dados de código python, fornecendo um auxílio com mais qualidade em requisitos para esta linguagem;

Code Llama - Instruct, uma variante ajustada para seguir melhor as instruções humanas, oferecendo uma interação conversacional mais aprimorada.

Treinados para processar sintaxe estruturada e múltiplos paradigmas de programação, os modelos CodeLlama representam uma alternativa moderna e de acesso aberto (open-source), ideal para cenários que exigem a conversão de requisitos em linguagem natural diretamente para código.

Além disso, temos também o StarCoder[3] e a sua evolução mais recente, StarCoder2[4], que representam modelos treinados em larga escala no dataset The Stack (The Stack v2 para o modelo StarCoder2), destacando-se pelo suporte multilinguagem, capacidades de preenchimento, janelas de contexto alargadas, permitindo, com este último ponto, analisar blocos mais extensos de código numa única passagem, e mecanismos de treino orientados para segurança e transparência. Graças a estas características, tornam-se referências úteis para compreender boas práticas na construção de LLMs orientados para código, tanto ao nível da arquitetura e dos datasets, como das técnicas necessárias para desenvolver sistemas capazes de converter descrições em código de forma fiável e coerente.

Para além dos modelos dedicados à geração completa de código, têm surgido ferramentas práticas de assistência ao programador integradas diretamente nas IDEs. Um exemplo marcante é o GitHub Copilot[4], estudado empricamente por Ziegler et al.[12].Baseado no modelo Codex, o Copilot funciona como um sistema de autocompletar código capaz de sugerir linhas ou blocos curtos com base no contexto imediato do ficheiro em edição. Ziegler et al.[12] estuda empiricamente a ferramenta analisando o comportamento da mesma e o seu impacto no processo de desenvolvimento, evidenciando como a completação incremental pode acelerar tarefas de rotina e reduzir o esforço cognitivo do programador.

estudaram

De forma semelhante, a Amazon apresentou o CodeWhisperer[5], analisado no respetivo estudo técnico[11], um assistente de programação concebido para gerar sugestões de código contextualizadas em múltiplas linguagens. Este trabalho compara o CodeWhisperer com outras ferramentas, incluindo o Copilot e o ChatGPT, avaliando a qualidade, correção e segurança das sugestões produzidas. Tal como o Copilot, o CodeWhisperer opera principalmente como um mecanismo de completação inteligente dentro da IDE, contribuindo para automatizar partes do fluxo de escrita de código e acelerar tarefas repetitivas.

Estas ferramentas demonstram que, para além de abordagens orientadas à geração integral de componentes de software, existe já uma adoção consolidada de LLMs em cenários de code completion no desenvolvimento real.

4 «Other Section(s) as Appropriate»

The report should include one or more sections providing a detailed description of the problem you are addressing in the project and your plan to tackle it. Use appropriate section titles for what is presented.

You should explain the methods you are planning to use, or have already started to apply, in your project. This discussion should be grounded in the related work, your own understanding of the problem, and, when available, preliminary results.

In case you already have some preliminary results, consider to include a section devoted to them. This section should describe the work already carried out, what data has already been collected, what analysis and designs have already been done, what methods have been used, what programs and/or preliminary results already exist, etc.

5 Forthcoming Work and Conclusions

This section should include subsections describing the work to be carried out during the remainder of the school year

[3]https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4]https://github.com/features/copilot

[5]https://aws.amazon.com/pt/q/developer/

and its objectives. It should also present a chronological plan for the completion of the project. Finally, include a concluding subsection that summarizes the contributions already made, provides a preliminary self-assessment of the progress achieved so far, and discusses the main difficulties encountered.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[2] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. 2024. The rising costs of training frontier AI models. arXiv:2405.21015 [cs.CY]

[3] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.* 2023 (2023). https://openreview.net/forum?id=KoFOg41haE

[4] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *CoRR* abs/2402.19173 (2024). arXiv:2402.19173 doi:10.48550/ARXIV.2402.19173

[5] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2023. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*. Springer, 387–402.

[6] Antonio Mastropaolo, Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto, and Gabriele Bavota. 2023. On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot. (2 2023). http://arxiv.org/abs/2302.00438

[7] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. *CoRR* abs/2308.12950 (2023). arXiv:2308.12950 doi:10.48550/ARXIV.2308.12950

[8] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *CoRR* abs/2402.07927 (2024). arXiv:2402.07927 doi:10.48550/ARXIV.2402.07927

[9] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). arXiv:2307.09288 doi:10.48550/ARXIV.2307.09288

[10] Yong Xu, Lili Bo, Xiaobing Sun, Bin Li, Jing Jiang, and Wei Zhou. 2021. image2emmet: Automatic code generation from web user interface image. *J. Softw. Evol. Process.* 33, 8 (2021). doi:10.1002/SMR.2369

[11] Burak Yetistiren, Isik Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *CoRR* abs/2304.10778 (2023). arXiv:2304.10778 doi:10.48550/ARXIV.2304.10778

[12] Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, X. Alice Li, Andrew Rice, Devon Rifkin, and Edward Aftandilian. 2022. Productivity Assessment of Neural Code Completion. *CoRR* abs/2205.06537 (2022). arXiv:2205.06537 doi:10.48550/ARXIV.2205.06537