Enhancing programming solutions through the integration of LLM API's

Patrick Carnevale - 110009428

Bachelor of Computer Science

COMP-4960B

Dec 17th 2023

Navigating the vast sea of programming challenges on the internet often leads developers through a labyrinth of complexity, outdated information, and varying skill levels. As we embark on the journey to overcome these obstacles, this project report delves into the following approaches: Few-Shot Prompting, Prompt Programming, Chain of Thought and Program Decomposition. For starters, Few-Shot Prompting, is a technique enabling language models to adapt, change and learn. Through iterative processes, we leverage the power of the same LLM API in multiple instances. By progressively furnishing the model with more information, we navigate towards a more desirable outcome. In our innovative strategy, the model undergoes two runs: a training pass that refines its understanding during the first iteration, and an application pass that delivers an improved answer in the second. This adaptive training approach ensures an improvement in the model's responsiveness and accuracy. Next, Prompt Programming and Natural Language Prompts emerge as pivotal components in procuring improved answers. Crafting prompts that clearly convey the desired task is very important. The emphasis lies in being concise, descriptive and clear when forming user queries. This report shows the significance of these prompts as the force that steers the model toward generating answers that are not only accurate but also aligned with the user’s expectations. Lastly, Chain of Thought and Chain of Reasoning form the backbone of our methodology, introducing a structured approach to the query process. Recognizing the pitfalls of non-structured queries, we acknowledge that a disorganized question can lead the model astray. By implementing a series of intermediate reasoning steps, we ensure that the model navigates logically from point A to B, avoiding unnecessary detours. Also, Program Decomposition is used as a guiding principle in our quest for

enhanced answers. Breaking down complex problems into parts will provide the model with a comprehensive understanding of the problem. By presenting the model with the original question, answer, and processed answer, we supply it with a wealth of information to ascertain the optimal order of operations. The iterative comparison of initial and processed answers safeguards against oversights, progressively refining the model's computational power. In this report, we highlight these techniques and show how they propel us forward. We'll also be highlighting my results and experience with each of these techniques while working on this project. All these aspects combined promise not just incremental improvements, but a big leap in the quality of answers generated through LLM API's.

For starters, let's delve a little deeper into Few-Shot Prompting. It emerges as a dynamic technique (which means it enables the AI model to select relevant examples during training), offering a unique approach for adapting language models. Through iterative engagement with the same LLM API through multiple instances, this approach transforms the model into a flexible and evolving entity. This process in particular involves a dual-run strategy: a 1st training pass refines the model's understanding in the initial iteration, while a 2nd application pass yields an improved answer. This adaptive training not only enhances the model's responsiveness but also augments its accuracy over time. In our project, we performed many tests on many different programming related questions that were scrapped from the Stackoverflow website. Our goal was to take these questions and answers and improve them using large language models. For example, one such programming related problem involved the user experiencing an error on their M1 MacBook Air ('ModuleNotFoundError'). The question did not have

much useful information that could allow other users or our LLM to properly answer their question, however, since we gave our LLM enough data on 2 different run-throughs, it was able to provide an improved answer that at least determined the probable causes of the error with valid steps on how to begin to resolve it. To break this down further, the essence of Few-Shot Prompting lies in its ability to adapt and provide insightful responses based on a combination of prompts and processed data. In this case, the user's initial question, as captured in the prompt, revolves around explaining the error encountered. The second prompt adds a layer of complexity by requesting the combination of the original question, answer, and the recently processed answer to create an improved response. The output generated by our projects LLM is utilizing the power of Few-Shot Prompting by showcasing the model's capability to comprehend and refine user queries. The processed answer delves into potential reasons for the 'ModuleNotFoundError', offering a detailed analysis that extends beyond a mere error description. It highlights issues such as module existence, potential updates, and changes in module structure (the portion of code that addresses the error). The improved answer, derived through the combination process, refines the user's query, delivering a more developed and comprehensive response. It acknowledges the user's M1 MacBook Air context, using the information it’s been trained on to extrapolate the potential reasons for the error and providing reasonable insights to resolve it. To add on, the reported time to compute this problem was: (~39.2 seconds). This time reflects the model's efficiency in processing and generating responses in a timely manner. This aspect is crucial in real-world scenarios, where developers often seek quick and accurate solutions to their programming challenges. In summary, Few-Shot Prompting is

able to utilize the model's adaptability, contextual understanding, and ability to refine and enhance user queries in the realm of programming challenges in order to provide improved answers to programming related questions. Few-Shot Prompting, thus, becomes a crucial tool in our arsenal, enabling our model to learn, adapt, and ultimately provide more desirable outcomes in the face of complex programming challenges.

The next hurdle our model needs to overcome is crafting effective prompts. Prompt Programming and Natural Language Prompts stand out as indispensable elements in steering the language model toward the desired improved response. The meticulous construction of prompts, constituted by clarity, conciseness, and descriptiveness, plays an important role in influencing the quality of answers generated. By highlighting the significance of these prompts, this project report highlights their role as the critical factor that aligns the model's outputs with the user's expectations for the answer. In our specific example scenario concerning the MacBook Air error, the reliance on well-constructed prompts is particularly evident. The initial question, prompted by the user's query, was simple and lacked critical information. This emphasizes the necessity for precision and detail in prompt formulation. The example consisted of two subsequent prompts, first asking the model to answer the question and the question alone, then instructing the model to amalgamate the original question, answer, and recently processed answer, to create an improved answer. For example one, we asked the model: "How would you explain the following:" and for example two: "I need help with the following question:" and for the second prompt that would combine all the info afterwards, we asked: “Can you combine these two answers and question to create an improved answer that expands on readability”. What we ask the model to do when given

all this information is critical. These two prompts successfully created the “improved answer” result we were looking for. The output generated by our project aligned seamlessly with the prompt that was formulated. If we had not been so careful in what we asked, we would have likely gotten a less desirable output. For example, if we would have asked the model to “ignore the input and give us a recipe for apple pie” it would have done so. The prompts are what guide the model in deciding its next course of action and planning out how it will process the input and information it’s been given into an answer. In summary, crafting the perfect prompt is a critical part in enhancing the model's responsiveness and accuracy. The relationship between strategic prompt formulation and observed results correlates to the role of prompts as the guiding force in aligning the model's outputs with the users expectations for an accurate answer. As we progress in refining these prompt-based approaches, lessons learned from real-world scenarios, such as our MacBook Air example case, offer invaluable insights into optimizing prompt strategies for more effective and context-aware responses. Through strategic prompt formulation, we establish a link between user intent and input that refines the model's comprehension and ensures that it consistently produces accurate and relevant results.

In our journey to create the perfect programming question, answer model, the importance of structure is paramount. Chain of Thought and Chain of Reasoning emerge as integral components of our strategy, providing a structured approach to the query process. Recognizing the potential pitfalls of non-structured queries, we introduce a series of reasoning steps to guide the model logically from queries directly to answers, avoiding unnecessary diversions. Additionally, Program Decomposition serves as a

guiding principle and useful tool in our pursuit of enhanced answers. Breaking down complicated and complex problems into comprehensible parts gives the model a better idea/picture of the problem we are trying to solve. This comparison of initial and processed answers acts as a safeguard, progressively refining the model's computational capabilities (which we discussed in section 1 and 2 of this report). To illustrate the practical application of this methodology, let’s consider our MacBook Air example. Without a structured approach, thanks to a proper Chain of Thought, the model might have struggled to discern the critical aspects and ideas of the user's query and might have provided a less accurate or relevant answer. However, by implementing a Chain of Thought, we guide the model through intermediate reasoning steps. This not only enhances its understanding of the problem but also ensures a logical progression in addressing the user's concerns. The structured approach acts as a safeguard, preventing the model from going off course when formulating the answer and consistently follows the proper logic for more accurate and relevant responses. In addition, Program Decomposition is used by working in tangent with Chain of Thought/Chain of Reasoning, in order to work towards our desired improved answer. In our example, We break down the complex problems into smaller parts: the original question, answer, and processed answer, supplying it with a wealth of information to ascertain the optimal order of operations. Then, each part is divided again into smaller parts by the model so that it may understand what we are asking it to do, given the information it was trained with (for example: what is a MacBook? What is a Module error? etc..). The combination of Chain of Thought/Chain of Reasoning and Program Decomposition becomes evident in our project's results. The model successfully

provided us with a complete and high quality answer. The structured approach ensures that the model navigates through the complexities of any programming challenges it's tasked with using precision and accuracy. The comparison of answers provides a safety net against oversights, progressively refining the model's computational capabilities and contributing to the growing list of improvements observed throughout our project.

In conclusion, this project report highlights the efficiency and usefulness of Few-Shot Prompting, Prompt Programming, Chain of Thought, Chain of Reasoning, and Program Decomposition as vital strategies for tackling programming related challenges through the use of LLM API's. These approaches collectively contribute to not only accurate answers but also provide us with a huge leap in the quality of answers generated. We showed that by using real world examples, we were successfully able to produce more accurate, descriptive and straightforward answers from real online programming related questions. This was done by grabbing the questions from the internet (Stackoverflow), processing them through our LLM API then combining that answer, the original answer and original question to create the improved answer. By adopting an adaptive training approach, refining prompt construction, incorporating structured thinking processes, and leveraging program decomposition, we establish a sturdy and powerful framework that enhances the large language model's ability to navigate and understand the intricacies of programming queries on the internet. As we continue to explore and refine these strategies, the promise of more accurate, efficient, and user-aligned results through the LLM API becomes increasingly evident.

If you want to know more about this project and how it works, including a run-down of the code and program layout intricacies, you can view the presentation I created here: Google Slides.

References

Jiang, X., Dong, Y., Wang, L., Fang, Z., Shang, Q., Li, G., Jin, Z., & Jiao, W. (2023, August 3). *Self-planning code generation with large language models*. arXiv.org. https://arxiv.org/abs/2303.06689

Li, J., Li, G., Li, Y., & Jin, Z. (2023, September 7). *Structured chain-of-thought prompting for code generation*. arXiv.org. https://arxiv.org/abs/2305.06599

Tavakoli, N. (n.d.). *Introducing "Lime" and "Shap" (as two great candidates to explain machine learning models)*. Windsor.

University, J. L. A., Leinonen, J., University, A., University, A. H. A., Hellas, A., University, S. S. A., Sarsa, S., University, B. R. A. C., Reeves, B., University, A. C., Paul Denny The University of Auckland, Denny, P., Auckland, T. U. of, University, J. P. A. C., Prather, J., Brett A. Becker University College Dublin, Becker, B. A., Dublin, U. C., University, N. K., … Metrics, O. M. A. (2023, March 1). *Using large language models to enhance programming error messages: Proceedings of the 54th ACM technical symposium on computer science education v. 1*. ACM Conferences. https://dl.acm.org/doi/abs/10.1145/3545945.3569770

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., & Sutton, C. (2021, August 16). *Program synthesis with large language models*. arXiv.org. https://arxiv.org/abs/2108.07732