Explainable prompts for LLMs

Submitted by:
Jafar Isbarov

Advisor: Prof. Stephen Kaisler

07.26.2024

1 Introduction

Large Language Models have seen a sudden rise in popularity in recent years. Their use cases include translation, summarization, and chatbots among other things. An often-cited drawback of these transformer-based models is their black-box nature. While they can be powerful tools for natural language, finding a connection between the input and the output can be challenging. This project is an attempt to solve this problem. We are building a simple and extensible method that highlights the parts of a prompt that affect the output the most.

For example, given the following prompt, a state-of-the-art LLM will probably give an advice about how to get help and take care of yourself:

What should I do if my husband shouts at me every day?

Our tool should highlight the words that affect this output the most. The main use case of such a tool is understanding LLM hallucinations, but it can also be used to optimize prompting strategies.

There is substantial literature on the explainability of LLMs [8]. There have been some works that attempt to solve this problem with counterfactuals. [6] creates counterfactual explanations for LLMs with a two-stage approach called MiCE. In the first stage, they train a new model specifically for generating counterfactuals. In the second stage, they perform a binary search to find the tokens that need to be edited (the actual counterfactual generation stage with the help of the trained model follows this). This is a powerful method that generates **minimal** and **fluent** counterfactuals. Minimal here refers to the number of edited tokens, while fluent refers to the fact that generated counterfactuals are intelligible to humans. The main limitation of this approach is that we need to train a new model for a new dataset (although it is possible that a trained model can be useful for other datasets as well).

[3] is another relevant paper that focuses on using counterfactuals to train a model that is robust to out-of-distribution (OOD) data. They generate counterfactuals with a method called CORE. This method works with the help of retrieval-augmented generation. While the MiCE method concentrates on the minimal and fluent nature of counterfactuals. CORE focuses on generating **diverse** counterfactuals. This is an elegant method if we have a large text corpus to work with, but that is not always the case.

While similar to these works, our project differs in an important aspect: We **can** use counterfactual explanations as a method, but our goal is highlighting the most important parts

of the original work, not generating counterfactuals. By highlighting the important words in the prompt, this method will allow LLM users to partially understand how the LLM produces its response. We have decided on several alternative approaches to this problem. They are discussed in the next section. As far as we are concerned, no previous work has attempted to build such a system.

2 Technical Approach

How do we identify the parts of the prompt that have the highest impact on LLM output? This is the main challenge of our project, and we propose two solutions. The first solution is called **removal method**. It is the simplest approach and will serve as the baseline. The second approach is called **replacement method**. We expect this method to function better, but it also comes with several open-ended questions.

Removal method This method works as follows:

1. Feed the original prompt to the LLM and record the output.
2. Iterate through the prompt:
    a) At each iteration, remove the current token and feed the new prompt to the LLM.
    b) Compare the new and the original output and record their difference.
3. Normalize the difference vector and use those numbers to color the text red. The higher the difference the more red the token should be.

The main challenge here is the comparison stage. How do we identify if the output has changed more? We have two options:

- Text embeddings. In this approach, we would vectorize the original and new outputs with an embedding model, and then find the similarity score of these two vectors. For example, we could embed the text with BGE Embedding models [2] and find the cosine similarity [7] between the vectors. This option will not work if we have similar text samples with opposite meanings.
- Entailment models. These models are useful for identifying whether two sentences are in agreement, contradict each other, or neutral [4]. The use of this method raises another question: Do we expect the changes to create a contradiction of the original response? Do we expect it to change the topic entirely? These questions need to be answered before the project can enter the final stage.

The replacement method can serve as a baseline for other methods, but it is too primitive to give reasonable results. There are a lot of words that carry little if any semantic weight, but replacing them changes the output dramatically. We can combat this issue by simply ignoring the stopwords.

Replacement method This is a simple extension of the removal method. Instead of removing the token, we replace it with another. The central question here is this: How do we find the new token to replace the original token with? We have multiple options here:

- Choosing a random token is not a reasonable solution, because LLMs tend to have a vocabulary size of 30-100 thousand.
- Replace with a similar token. In this case, how do we find similar tokens? We could use embedding or entailment models. In the case of embedding models, we can cache the embeddings for all tokens and later search through them. For entailment, models, we would need to calculate a matrix of shape $V * V$, where V is the vocabulary size.
- Replace with an opposite token. How do we find opposite tokens? Even if we use embedding or entailment models, finding words of opposite meaning is a completely different problem from finding similar words.

Stopwords In the early stages of development, we discovered a new challenge: A considerable portion of the words in a prompt are clearly not important. Testing by removal or replacement methods costs us time and money. We need a way to identify and skip over these words. We are in the process of integrating NLTK stopwords [1] into our pipeline to achieve this. With this addition, we expect the output to be as follows:

What should I do if my husband shouts at me every day?

As you can see, stopwords have been greyed out, and not considered during the analysis.

Tools of the trade Python is used to develop the method, develop both the client and the server, and connect to LLMs and the database. More specifically, we use FastAPI to develop the server and Streamlit to develop the client. SQLAlchemy is used for the database connection and various client libraries are used to connect to the LLM APIs. SQLite is used as the database since we have no scalability issues. Together AI's Embedding API is used since we have no hardware to run an embedding model efficiently. OpenAI API is used for the GPT models. We also use ollama occasionally to run smaller models locally. This helps us keep the cloud bills small. Another such alternative is the Transformers library by Hugging Face, which helps us run certain embedding models locally with minimal coding.

We have used Docker for containerizing the application, Docker Compose for orchestrating multiple containers, GitHub for hosting the code, GitHub Actions for creating CI/CD pipelines and DigitalOcean for deployment.

3 Software Architecture

After developing the method, we need to think about building the software around it and integrating it with other software.

The system consists of four main components: client, server, LLM, and database. The client interacts with the user and the server to create a connection between them. The server accepts the prompt from the user and executes the main business logic that results in a highlighted prompt, which is sent back to the user. A large language model is necessary for the server to run and analyze the prompt. We also use a database to log the interaction with the user. You can see the high-level architecture in Figure 3.1. Every time a prompt is received from the user, the server starts generating perturbed prompts and sending them to the LLM provider. After all results are ready, the difference scores are calculated and sent back to the frontend. Before this happens, the results are logged into the database. You can see this flow in Figure 3.2.

Client Client is developed with the help of Streamlit[1]. We have a simple UI that allows the user to write a prompt and returns its highlighted version. There is also a limited number of LLM options to choose from. UI does not only show the user the highlighted prompt but also allows them to download the data used to highlight it. This data contains a score for each token in the prompt.

Server The server side is more complicated. It deals with the database to log the interaction with the user. We do not support accounts and we have no way of differentiating between different users. The server is developed with the FastAPI framework since it is (1) easy to use and (2) supports asynchronous communication. Since we deal with multiple LLM providers, backend contains multiple LLM clients that are called on demand.

Database An SQLite instance is deployed alongside the Streamlit application as a log database. It interacts with FastAPI via SQLAlchemy. This is an object-relational mapper that prevents SQL injection attacks. We log to the database the prompts sent by the users, and all communications with the LLMs, since they are too expensive. We use these records as a cache to decrease the cost of LLM APIs.

[1] `https://streamlit.io/`

LLM In principle, any LLM can be used as a part of this system. The main challenge is that different cloud services come with different interfaces. This can add extra development time. We offer several models that the user can choose from, and adding new models is possible on demand. These models are:

- GPT-3.5 by OpenAI
- LLaMA 2 7B by META AI (through Together AI)
- Mistral 7B by Mistral (through Together AI)

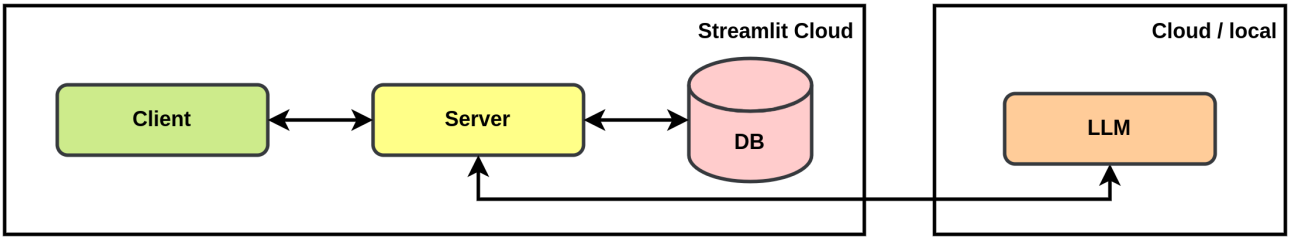

Figure 3.1: High-level architecture of the system. Client and server are built and deployed with Streamlit, alongside an SQLite instance. LLM is an external component.

Wherever possible, instead of sending multiple requests to the LLM provider, we will send a batch request. This requires that all perturbed prompts should be ready before the API call is made. This means that we cannot use a perturbation method that has a sequential dependency on previous prompts.

4 Data

Developing a new method is not sufficient. This method must be tested against a dataset of considerable volume and variety. We decided to use three different data sources: the RACE dataset, **MohamedRashad/ChatGPT-prompts** dataset and manually curated questions from research papers.

The RACE is a reading comprehension dataset with more than 28,000 passages and nearly 100,000 questions [5]. Its questions can be used as sample prompts.

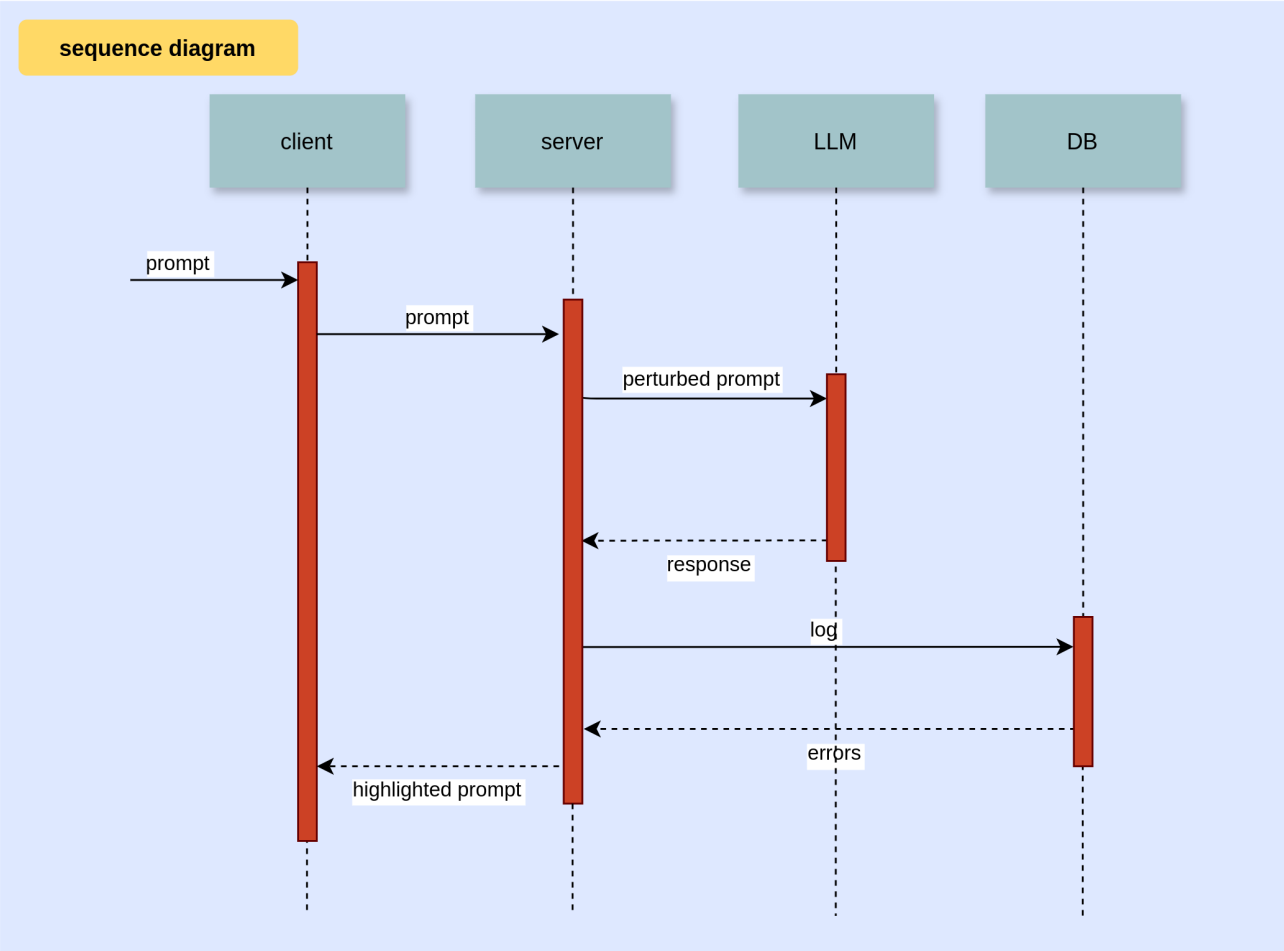

Figure 3.2: Sequence diagram of our application. The server interacts with the client, LLM, and the database to manage the entire process.

Human prompt	**ChatGPT response**
What are the causes and effects of strict parents?	Strict parenting can have a variety of causes, such as a desire for...
What should I do if I'm feeling depressed?	If you are feeling depressed, it is important to seek help and support...
Please write a breaking news article about a leaf falling from a tree.	Breaking News: Leaf Falls from Tree. In a shocking turn of events...

Table 4.1: Some samples from MohamedRashad/ChatGPT-prompts dataset.

MohamedRashad/ChatGPT-prompts dataset is a collection of ChatGPT prompts and their responses. We can use it in our removal and replacement experiments. Since the original response is available, we would save money by decreasing the number of API calls. However, since the model name is not mentioned in the dataset, it is possible that we would be using

a different model or a different model version, resulting in faulty output. Some samples from this dataset are available in Table 4.1.

Manually curated questions constitute a smaller dataset, but they allow us to test the edge cases better.

5 Conclusion

Our main goal is to develop a tool for highlighting the most important parts of an LLM prompt. This tool should be (1) fast, (2) simple, and (3) extensible to other models. We achieve this by creating two alternate solutions: removal and replacement methods. The former serves as the baseline while the latter is our main attempt at a solution.

During the development of these methods, we used the RACE dataset and a hand-picked list of prompts from research papers.

The software architecture is a traditional three-tier web server, with the addition of an external LLM provider. It was developed with Streamlit for the frontend, FastAPI for the backend, and SQLite for the database.

Bibliography

[1] Steven Bird and Edward Loper. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[2] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2024.

[3] Tanay Dixit, Bhargavi Paranjape, Hannaneh Hajishirzi, and Luke Zettlemoyer. CORE: A retrieve-then-edit framework for counterfactual data generation. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2964–2984, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[4] Daniel Z. Korman, Eric Mack, Jacob Jett, and Allen H. Renear. Defining textual entailment. *Journal of the Association for Information Science and Technology*, 69(6):763–772, 2018.

[5] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[6] Alexis Ross, Ana Marasović, and Matthew Peters. Explaining NLP models via minimal contrastive editing (MiCE). In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3840–3852, Online, August 2021. Association for Computational Linguistics.

[7] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24:35–43, 2001.

[8] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey. *ACM Trans. Intell. Syst. Technol.*, 15(2), feb 2024.

Appendix A: Revisions

The following changes have been made to the original proposal:

1. The original topic concentrated on the mechanistic interpretability of neural networks, but I am now working on explaining the model through their I/O behavior.
2. According to the original proposal, I was supposed to work with small neural networks to decrease the computational load. I switched to LLMs and decided to use APIs to handle the issue of hardware.
3. The original deliverable was a Python package. I switched to a website instead to make the application more accessible. The codebase will still be publicly available.