Google Summer of Code 2024 - Project Proposal

LLM Assistant for OpenROAD

Model Architecture and Prototype

Palaniappan R

palaniappan.r.mail@gmail.com

Abstract

This project aims to develop a conversational chatbot built around OpenROAD, designed to answer user queries. The chatbot should be able to assist both beginners and seasoned developers with using OpenROAD tools.

My goal is to build a complete pipeline that includes dataset updation, model training, and model deployment. This will be a scalable architecture that can adapt to evolving datasets with ease, and also integrate metrics to regularly keep track of the model's performance.

Introduction

The OpenROAD Project

OpenROAD aims to create an autonomous, open-source layout generation flow to democratize semiconductor development. It provides a seamless RTL to GDSII capability for cost-effective design exploration and tapeout. The goal is a 24-hour, NHIL (No Human In Loop) design process with zero PPA (power-performance-area) loss.

I have been using OpenROAD and ORFS for the past few months, and as a new user, here are solutions to some pain points I experienced.

Help with Installation Related Issues

During my first installation of OpenROAD, I discovered several discussions to solve basic installation issues on GitHub and Slack channels. These queries are often simple to address and can be solved without the intervention of community members. One of the primary goals of this project is to help the end user resolve issues of this sort. Furthermore, several users ask queries about supported operating systems and Docker images on public forums, this can be addressed using this project as well.

Provide Troubleshooting Assistance

As a part of this project, an automated pipeline that pulls closed issues from the OpenROAD and OpenROAD flow scripts GitHub repositories will be built. Once this pipeline is set up, end users can query the LLM to resolve recently identified issues, without having to read through GitHub issue threads. The model will have the ability to summarize the thread and provide end users with actionable information.

Easier Access to Existing Resources

The extensive and detailed documentation of OpenROAD and OpenROAD flows can be overwhelming to new users seeking information on specific topics. As an alternative to navigating the entire documentation, our model will provide a summarized solution to the user query. Additionally, it will provide a link to the source of the summarized information. This is possible by the RAG (retrieval augmented generation) setup proposed here.

Primary solutions to these pain points have been addressed in the initial prototype I've built (more on this here)

Project Goals

1. Project Objectives

The main objective is to develop an LLM assistant tailored to assist beginners and seasoned developers with using OpenROAD tools. I aim to build upon existing work (done during GSoC 23') and design a working prototype.

My goal is to build an application that is able to assist experienced users and beginners alike. The final deliverable will be an LLM pipeline that can be used with either locally run open-source LLMs or with public LLM APIs.

The target user base for this project can be divided as follows,

a. Hardware designers and developers familiar with the OpenROAD project
b. Beginners starting out with OpenROAD

The initial objective is to reduce repeated and redundant queries about installation on public forums such as Slack channels and GitHub discussions.

Primary Objectives

- Help with installation issues, with regard to both OpenROAD and OpenROAD-flow-scripts. The bot can help users with troubleshooting and guide them to specific installation methods, like a prebuilt binary, if a build from source repeatedly fails. The bot will be able to answer queries regarding available prebuilt binaries, currently supported operating systems and recent Docker images.
- Address a comprehensive list of commonly asked questions. (there is more detail on this later in this document)

Once an initial prototype is built, I will move to integrating more complex use cases, which shall benefit a wider target demographic, including hardware designers and developers.

Additional Use Cases

- Generate platform configurations, design configurations, design goals and timing constraints for various designs.
- View and summarize logs and results (for area, power etc.) in various stages of the RTL - GDSII flow
- Help with general troubleshooting by parsing through GitHub issues and documentation.

2. Expected Deliverables

- A robust LLM pipeline that integrates existing documentation, the new man-pages docs and online resources detailing the OpenROAD project. The pipeline could be run locally using open-source LLMs or using public LLM APIs.
- Open Source fine-tuned model weights for users to download and use.
- A deployed web-backend that offers access to the LLM. Additionally, the model can be made accessible with existing OpenROAD documentation and OpenROAD-flows pages.
- Proper documentation for the implemented architectures.
- Weekly progress reports

3. Proposed Stretch Goals

- Integrate a separate version of the application with the OpenROAD QoR Dashboard, designed to assist more advanced users with their workflows..
- Integrate the model with the OpenROAD GUI.
- Explore the use cases of a VLM (Vision Language Model) with the OpenROAD GUI
- Build a full fledged chat assistant designed to help write scripts with OpenROAD tools, something similar to this ChatEDA paper.
- Build a separate website designed to educate beginners about OpenROAD flows. We aim to achieve this by including the docs of all various tools used by OpenROAD (like Yosys, ABC, OpenDB etc.) The LLM will be built to be able to answer questions about any step in the RTL - GDSII flow.

Current Implementation

Over the past few weeks, I have been building a prototype architecture for this project.
The colab notebook for the same can be found here: OpenROAD - PoC 2 - Ensemble.ipynb
Here are some features of the architecture in my current PoC,

1. Dataset Embedding and Chunking

Datasets, including OR docs, ORFS docs, OR user guides, OR issues and OR discussions, are embedded into FAISS vector databases. Currently, embedding models from Hugging Face (HF) and Google's Gemini Embeddings models are being used, with plans for fine-tuning on one of these models in the near future.

Currently, there are three main data sources,

1. OpenROAD docs + OpenROAD flow-scripts docs (info about installation, etc.)
2. OpenROAD User Guide (info about OR's individual modules and commands)
3. OpenROAD GitHub Issues + GitHub Discussions (q/a conversations)

Each of these data sources is then embedded into a separate FAISS vector database.
Given the nature of these data sources, I found the quality and relevance of the retrieved documents to be significantly better than embedding all the data sources into a common vector database. But in the future, we can consider generating a synthetic dataset comprising multiple data sources.

Prior to embedding, documents are recursively chunked to enhance granularity.

2. Retrieval and Reranking Process

Upon query, the system queries all three vector databases to retrieve relevant documents (those that were previously chunked). The retrieved documents undergo reranking utilizing the Reciprocal Rank algorithm. Langchain's Ensemble Retriever facilitates this reranking process.

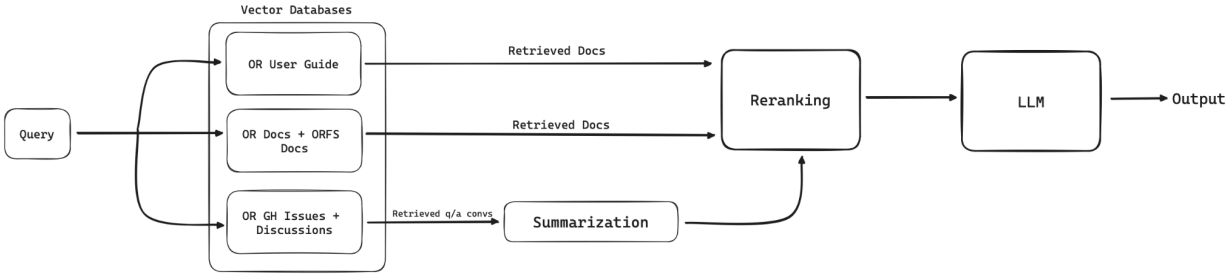
After retrieval and reranking, the documents retrieved from the docs and user-guide vector databases are fed directly to the LLM.

The documents retrieved from the q/a datasets are first summarized (by the LLM itself) and then fed back into the LLM as context. I found this approach to enhance the LLM's inference capabilities, when compared to directly feeding in a q/a conversation.

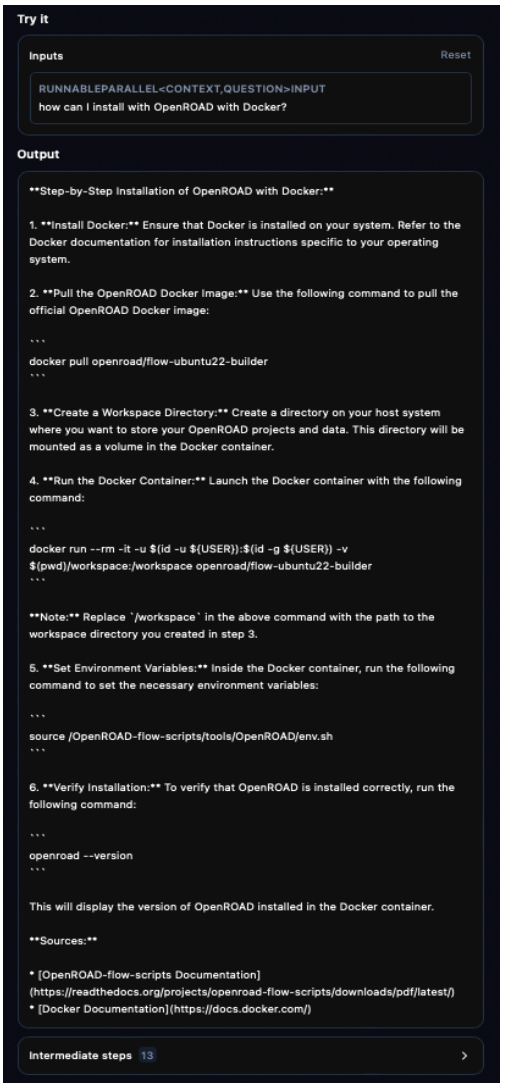
3. Model Execution

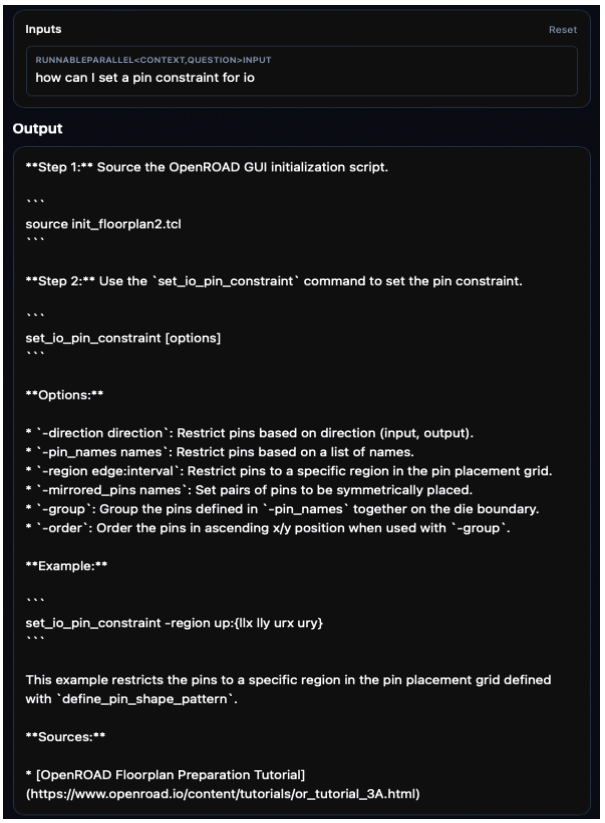
The reranked documents serve as input context for the LLM to answer questions. In the notebook, you can choose to either use Google's Gemini Pro or Mistral-7B-Instruct as your LLM.

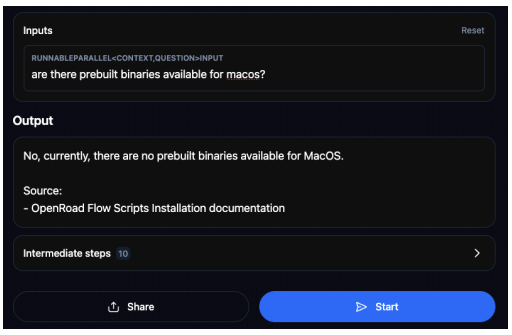
My current architecture,

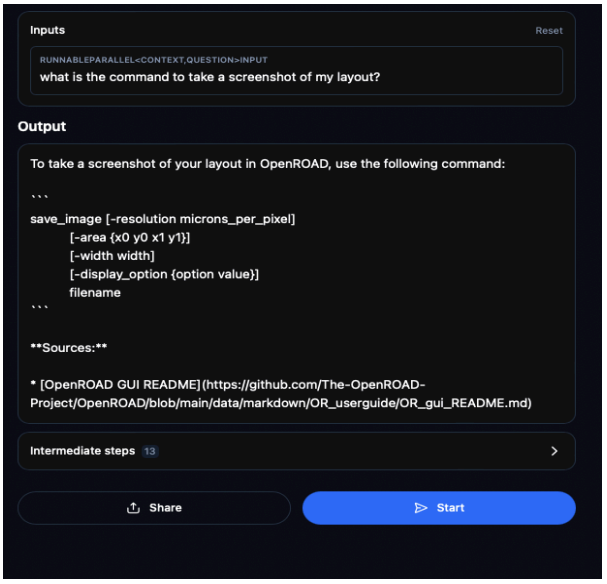


Here are some example prompts fed to my current prototype, (more examples here: 🗀 OR-LLM-demo)









Implementation Plan

1. RAG Architecture

Currently, this is how the pipeline works, (this is illustrated in my architecture's flow chart as well)

1. Query gets inputted into the LLM.
2. The query will be converted into an embedding and provided to a search function.
3. The search function compares the query with vectorized documents in the different vector databases and retrieves relevant documents.
4. The retrieved documents are reranked and fed back into the LLM, which will run text-generation and craft an appropriate response to the query.

In the initial few weeks, I plan to look at more advanced RAG architectures and compare their performances with each other.

2. Retriever Methods

As mentioned earlier, I found significantly better results with recursive chunking. Right now, before retrieval, the documents are chunked with 500 tokens first, followed by chunking with 256 tokens. I will be choosing the optimal chunking parameters, and exploring embedding models with suitable context lengths accordingly.

Here are some possible retrieval architectures,

Ensemble Retriever

The ensemble retriever is ideal for use-cases with multiple data-sources. Currently in my initial PoC, I am using an ensemble retriever with three separate vector stores, as mentioned above.

The ensemble retriever uses the reciprocal rank algorithm, to re-rank the documents retrieved from the three different vector stores (OR userguide, OR/ORFS docs, OR GH issues/discussions q&a). The documents retrieved from the OR GH q&a dataset are summarized before being used for answer generation by the LLM.

The ensemble retriever considers a user defined weight assigned to each vector store, for better retrieval. Currently the userguide dataset has been given the highest weight, 0.5, with the docs and q&a datasets getting 0.3 and 0.2 respectively. I am looking to build a system that automatically assigns the weights depending on the user's question.

Multivector Retriever

The multivector retriever adds more complexity to some of the features in my current architecture, like smaller chunking and document summarization. This architecture employs LLMs to generate hypothetical questions for each document, and embeds them alongside the document.

Essentially, this architecture aims to embed each document as multiple vectors (the document's summary, q/a pair etc.) in the vector store, to facilitate more accurate retrieval.

Self-Querying

This method uses a separate LLM chain to craft unique queries for each prompt. The LLM crafted queries are then fed to a vector store for document retrieval. Given the number of LLM calls for each prompt, this architecture might prove commercially infeasible for hosted public LLM APIs. However, depending on the performance I plan to consider this in cases where a local LLM is used.

3. Dataset Curation + VectorDB

Currently Available Resources

In my initial PoC, I am using a dataset comprising,

- OpenROAD userguide
- OpenROAD readme doc
- OpenROAD documentation
- OpenROAD-flows tutorials
- Resolved OpenROAD GitHub Issues
- Resolved OpenROAD flows GitHub Issues

I plan to integrate these sources as well,

- Documentation for Yosys, ABC, Klayout and other tools used by OpenROAD
- Slack channel message threads (Jack mentioned this in his project report here)

I plan to work with the dataset curation team (which in itself is a separate GSoC project) and build a better dataset suited for RAG.

A pipeline that pulls READMEs and GitHub Issues from the OpenROAD repository has already been built. In the initial phase of the project, I will work on extending this to the OpenROAD-flow-scripts project.

Synthetic Dataset Generation

Currently available options for synthetic dataset generation,

a. Custom prompts can be designed to generate Question/Answer pairs based on our text corpus. We can employ state of the art models like Gemini Ultra for this purpose.
b. The Ragas Framework constructs a Q&A dataset from an existing source of information. Thus, we can use our current knowledge base (consisting OR docs, ORFS docs, GH Issues, etc..) to generate a massive Q&A dataset that can either be used for RAG or finetuning.
c. There are recently published research papers, like Promptagator ,about building synthetic datasets for RAG. If time permits, I shall consider this as well.

In the dataset I used for my initial PoC, certain question-answer conversations contain extensive output logs and tables. This could potentially be affecting the semantic search algorithm used for document retrieval, I plan to generate synthetic data for such documents.

Exploring Chunking Techniques

Depending on the nature of the RAG knowledge base we are working with, novel document chunking techniques like recursive chunking improve the quality of the segmented documents. This in turn improves the relevance and granularity of the retrieved snippets. In my initial PoC, I noticed a significant increase in performance, after chunking my documents recursively.

Finetuning Embeddings

A popular method to improve the performance of our RAG bot is to finetune the embeddings used for our vector database. This finetuning aims to group semantically similar pieces of information together, to improve general retrieval performance. Once the initial dataset is finalized, an open-source embeddings model can be finetuned and saved for use. To finetune embeddings, we first generate a small synthetic dataset from the text chunks, consisting of relevant questions and answers . Then a small embeddings model like `MiniLM` or `BGE-small` will be finetuned using the `sentencetransformers` API. Different synthetic datasets and models will then be compared and evaluated against each other.

4. Model Finetuning

After finalizing our RAG pipeline, we should consider fine-tuning our LLM for better answer generation. A fine-tuning dataset will be built and tested on smaller open-source LLMs first. Once we see significant improvements in our fine-tuning + RAG setup on smaller open-source LLMs, we should move to an LLM with API access (if necessary). As we have discounted access to Google Gemini, we could consider using this as well. (Finetuning Gemini Pro was made available recently, https://developers.googleblog.com/2024/03/tune-gemini-pro-in-google-ai-studio-or-gemini-api.html)

Following this, to boost the reasoning capabilities of the LLM, I will look at some recent research papers in this field. Some of these include,

- ReAct
- Chain-Of-Thought
- Tree-Of-Thoughts
- Multimodal Chain-Of-Thought

5. Evaluation Metrics

These RAG architectures come at the cost of both compute power and latency. Thus a thorough overview of all the possible architectures must be done, and their efficacy, latency and other metrics must be compared before choosing the best architecture for our use case.

When we iterate through different RAG Architectures, we will need some evaluation metric to assess each architecture, here are a few possible options,

Automated Evaluation

a. The RAGAS framework
   This is a very recent framework developed exclusively for evaluating RAG pipelines. It includes a lot of metrics like Context Relevancy, Context Recall, Faithfulness, Answer Relevancy, Answer Correctness etc. However, we will need a proper dataset to use RAGAS on.

b. ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
   This is a metric commonly used for testing the summarisation capabilities of an LLM. This will be a crucial metric for our use case, given how we want our model to parse through the documentation and give precise answers for basic questions.

c. BLUE, BLUERT
   These metrics help quantify the machine translation capabilities of our model, comparing the generated text and the reference text.

d. METEOR (Metric for Evaluation of Translation with Explicit ORdering)
   This is another metric, designed to quantify machine translation.

Human Evaluation

Have a small dataset (30 - 50 questions) about OpenROAD. An expert human evaluation team can rank different LLMs and architectures based on scores given to the questions in the dataset.

The quality of this dataset is paramount to our application. A custom pipeline shall be set up, that runs this dataset through different iterates of the model architecture, to compare and benchmark them along the way.

Benchmarking Pipeline

Once the standard dataset for evaluation has been finalized, I plan to set up a complete pipeline to compare and benchmark different RAG architectures and finetuned models.

There are several open-source tools that help with this,

- LangSmith
- LLMPerf
- LangChain Evaluators

6. Web-UI & Docs Integration

After finalizing the finetuning and the RAG architecture, I will start working on building a web backend that serves the LLM. This backend can be hosted on platforms such as AWS. (more detail on this in the Deployment section).

A simple user-friendly frontend will be written with JavaScript frameworks (such as React) and made available for hosting.

Additionally, after discussions with the OpenROAD development team, I will work on making the LLM directly accessible on the documentation pages of OpenROAD and OpenROAD-flow-scripts. Recently, many open-source projects have begun integrating chatbots into their documentation pages, an example is Langchain (https://chat.langchain.com/).

7. Deployment

Once the complete pipeline has been built, I will proceed to setting up CI/CD (Continuous Integration/Continuous Deployment) workflows for this project. The goal is to automate the data extraction, training and model deployment process, to keep the system operational with minimal intervention.

The model will either use an open source finetuned model or a fine tuned LLM API and will be hosted on platforms like AWS. With reference to the scalable architecture mentioned in the abstract, this pipeline will be able to automatically fetch data, finetune (if required) and deploy the model automatically.

Once we have a list of resources the initial dataset was built with (OpenROAD documentation pages, GitHub Issues pages, Slack channels), the proposed pipeline will be able to automatically scrape the websites for resources. Thus, the dataset for RAG gets continuously updated at regular time intervals.

After each update/finetune, the pipeline will run the modified architecture through the evaluation metrics. This will help us compare different versions of our architecture against each other.

8. Plan for stretch goals

The proposed website will walk users through each step mentioned in the OpenROAD flows tutorial page. This website can be hosted separately and the LLM can be accessed via an API or hosted on platforms like AWS Bedrock.

The model can help users navigate through and use different views the GUI offers, like Placement Density, Power Density, Routing Congestion etc.

- The LLM could point out regions of less/more routing congestion.
- Turn on/off different layers on command
- Select specific objects and give information about the selected object.
- Help with general investigation of the design.

Project Timeline

Community Bonding - May 1 - May 26

- Do further research.
- Setup a github repository for the project and Google Collab environments for testing.

Week 1 - May 27 - June 2

- Curate the dataset that is to be used for model evaluation.
- Complete benchmarking pipelines for evaluating the built RAG architecture.
- This will be used in the coming weeks to evaluate and compare all the different architectures.

Week 2 - June 3 - June 9

- Work on integrating the new manpages poc into the RAG architecture.
- Test and compare different chunking methods and RAG architectures.

Week 3 - June 10 to June 16

- Explore synthetic dataset generation.
- Work on fine-tuning an open source embeddings model.
- Research methods to better rerank documents, optimizing pre/post retrieval.

Week 4 - June 17 to June 23

- Explore model finetuning.
- Create a custom dataset for finetuning a smaller open-source LLM.
- Find a suitable model to test the finetuning on.

Week 5 - June 24 to June 30

- Host the application and enable forms for testing and user feedback
- Write documentation for helping users set up the LLM pipeline locally.
- Explore LLM API access and hosting.

Week 6 - July 1 to July 6

- Setup LLM deployment infrastructure.
- Buffer week before the midterm evaluation.
- Work on midterm evaluation report.

Week 7 - July 7 to July 13 - **MIDTERM**

- By the midterm, my goal is to have a working application that is hosted online.
- Open the application for testing and gather feedback from OpenROAD devs and users.

Week 8 and Week 9 - July 14 to July 27

- Improve the RAG architecture, based on the collected feedback.

- Begin working on a separate pipeline where the LLM educates beginners about the complete RTL - GDSII flow. The application will guide users through the entire process and answer any questions they have along the way.

Week 10 - July 28 to August 3

- Write documentation for the tested architectures and curated datasets

Week 11 - August 4 to August 10

- Buffer week + time to work on stretch goals. Work on having LLM automatically generate scripts for platform configurations, design configurations, design goals, and timing constraints for various designs.

Week 12 and Week 13 - August 11 to August 25

- Work on integrating the LLM with the OpenROAD documentation page.
- Work on the final GSoC Report

My Biographical Information

Basic Information

Name: Palaniappan R
Major: Electronics and Mathematics
Year of Study: III
University: Birla Institute of Technology and Science, Pilani, India
Personal Email: palaniappan.r.mail@gmail.com
University Email: f20212915@hyderabad.bits-pilani.ac.in
GitHub: https://github.com/palaniappan-r
Slack: https://osre2024.slack.com/team/U06LZKEKFRP
Resume: Palaniappan R - Resume.pdf

About Me

I am currently an undergraduate pursuing a bachelor's in Electronics and a master's in Mathematics from the Birla Institute of Technology and Science, Pilani. I have significant experience working with LLMs and building scalable web infrastructures.

Recently, I had the opportunity to work on a LLM powered chatbot. The chatbot had to have the ability to engage in multiple conversations concurrently with different users. I developed a framework that keeps track of conversational memory and context separately, across different users and conversations. The goal was to provide a more personalized experience to each user.

In summary, this details the work I've done,

1. Extensively tested various open-source LLMs from HuggingFace.
2. Using Langchain, I set up a custom RAG framework, where the LLM was prompted multiple times using various data sources (for each user input), to get the desired results.

3. Built a small dataset and created custom automatic pipelines that embed the dataset into a vectorDB, at specific intervals when the dataset gets updated.
4. Developed custom functions to help the LLM keep track of the conversation memory separately for each user in parallel, leading to better in-context responses.

The application was required to be built at scale, thus I had to optimize the chat pipelines to ensure quick response with reasonable latency.

A basic implementation of the LLM server I mentioned above can be found here, on a public GitHub repository.

I have significant experience with web development as well. I completed an internship at UST Global over the last summer and built scalable microservices. I am an avid supporter of open-source projects and contributed significantly to the development of an open-source cab-sharing application used on campus.

Miscellaneous Information

The GSoC project timeline perfectly aligns with my summer break. Thus, I am confident about my ability to devote the time required for this project. I hope to effectively communicate with my mentors via email/Slack and schedule meetings that align with their respective time zones.