Internship Report - Generative AI

Sanath Vijay Haritsa
11038004
SRH Hochschule, Heidelberg
April 4, 2025

Schwarz IT - Creative Text Team

Schwarz IT is the IT backbone of the Schwarz Group, which includes Lidl and Kaufland, making it one of the largest retail IT service providers in Europe. The company focuses on cloud computing, cybersecurity, AI, and enterprise solutions to support retail operations at scale. The company is leveraging AI-driven automation to enhance personalization and operational efficiency across various domains, including HR, recruitment, customer experience, and intelligent retail solutions.

This team primarily focuses on text generation use cases, such as optimizing product descriptions for online retail, generating technical user manuals, and providing translation services. Currently, many of these tasks are outsourced to external agencies, costing the company millions of euros annually and often leading to delays due to lengthy processes. Developing an in-house, automated system can streamline operations, reduce reliance on third-party providers, and minimize human error.

1 Task - LLM Evaluation without Ground Truth

1.1 Introduction

Evaluating text generated by Large Language Models (LLMs) like GPT can be tricky, especially in the absence of a "ground truth" or a definitive correct answer. This situation often arises in creative tasks, open-ended dialogues, or areas where multiple valid outputs are possible.

In this task, multiple large language models (LLMs) are employed to answer questions based on a conversation between two individuals. The goal is to evaluate the quality of the generated answers and determine which response and model perform the best. Since the conversation may not provide a single "correct" answer, evaluation will rely on semantic similarity and retrieval techniques to assess the relevance of the generated answers.

By leveraging retrieval techniques, the system can retrieve the most relevant documents or parts of the conversation and use these as reference points to evaluate the generated responses. The retrieved documents can help establish a semantic baseline for comparison, offering insight into whether the model's response stays relevant to the core themes and topics discussed in the conversation.

1.2 Literature Survey

1.2.1 Ranking Large Language Models without Ground Truth

Dhurandhar et al. (2023) introduced the G-MM framework for evaluating large language models without predefined correct answers. The framework employs a Generator (G) to create responses from multiple LLMs, which are then ranked by a Meta-Model (MM) based on criteria such as fluency, coherence, and relevance. The MM ranks models by aggregating pairwise comparisons, without knowing which model generated each response.

Their experiments show that G-MM aligns well with human judgment, is robust to different generators, and scalable across various criteria. While it acknowledges challenges like bias and prompt dependence, the framework marks a significant step toward automated LLM evaluation. To apply G-MM to conversational QA, evaluation criteria like coherence, consistency, fluency, relevance, appropriateness, and conciseness are defined. Diverse conversations are paired with questions, and multiple LLMs generate responses. Pairwise comparisons of these answers are evaluated against the criteria, with scores aggregated to rank the models. This method provides valuable insights into model strengths and weaknesses, with continuous refinement of evaluation criteria for improved assessments.

1.2.2 Can Large Language Models Be an Alternative to Human Evaluation?

Cheng-Han Chiang and Hung-yi Lee investigate the potential of using large language models (LLMs) as substitutes for human evaluation in natural language processing (NLP) tasks. The authors conducted experiments on three tasks—text classification, question answering, and summarization—using a multi-faceted approach to assess the effectiveness of LLMs in replacing human evaluation.

They compared the performance of pre-trained LLMs (such as BERT and RoBERTa) and self-supervised LLMs to human evaluation, using metrics like accuracy, F1-score, and ROUGE score. The study also included a human evaluation component where annotators rated the quality of the LLM-generated outputs. The results indicated that LLMs could reliably replace human evaluators, particularly for tasks requiring extensive human annotation. Pre-trained LLMs outperformed self-supervised ones, and LLMs were shown to help identify high-quality outputs and provide valuable feedback for human annotators, improving the overall quality of evaluations.

1.2.3 Survey on LLM-as-a-Judge

Gu et al.'s "Survey on LLM-as-a-Judge" explores using Large Language Models as automated judges, offering a potential alternative to subjective and costly human evaluation. The survey systematically analyzes existing research, defining "LLM-as-a-Judge" as using LLMs to evaluate content quality. Through literature review and categorization based on judgment tasks, LLM architectures, prompt engineering strategies, and evaluation metrics, the authors highlight the field's potential. Key challenges identified include bias, explainability, robustness, and generalization, emphasizing the need for further research in these areas. While acknowledging the promise of LLMs in automating judgment tasks, the survey stresses the importance of addressing these challenges, particularly through exploring human-LLM collaboration, to develop more robust, fair, and transparent LLM-based judgment systems.

1.3 Data

The evaluation is conducted using three simulated conversations between two people, each focused on a specific topic: "tourism in Europe," "Apple and its products," and "global warming." From these conversations, a total of 15 questions are generated.

1.4 Methodology

1.4.1 Conversation Embedding and Storage:

The first step involves storing the conversation between two people in a vector database. To achieve this, the conversation is passed through an embedding model to transform the text into numerical vector representations. These embeddings capture the semantic meaning of the conversation in a dense format, allowing for efficient semantic search and comparison. Once the embeddings are created, they are stored in a vector database, which allows for fast retrieval based on similarity.

1.4.2 Semantic Search to Retrieve Relevant Documents:

When a new question is posed based on the conversation, the system performs a semantic search over the conversation embeddings stored in the vector database. Semantic search relies on similarity to measure the closeness between the query and stored documents. The top 3 most relevant documents are retrieved based on the similarity between the query embedding and the conversation embeddings.

The key here is that the search is semantic, meaning that even if the retrieved documents do not share exact wording with the question, they can still be relevant if their meanings align.

1.4.3 Answer Generation Using Multiple LLMs:

Multiple LLMs are used to generate answers to the query. The idea behind using multiple models is to capture the variability in responses from different models, as each might interpret or express the answer in a different way. Each LLM is given the same prompt (the question, along with the relevant context or conversation) and generates an answer accordingly.

1.4.4 Embedding the Generated Answers:

Once the LLMs generate their answers, these answers are passed through the same embedding model used for the conversation. This step ensures that the answers are represented as dense vectors, allowing for an easy comparison with the relevant documents retrieved earlier.

1.4.5 Cosine Similarity Calculation

The next step involves calculating the cosine similarity between the retrieved documents and the generated answers. Cosine similarity is a measure of similarity between two vectors by calculating the cosine of the angle between them. The formula for cosine similarity is:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Where:

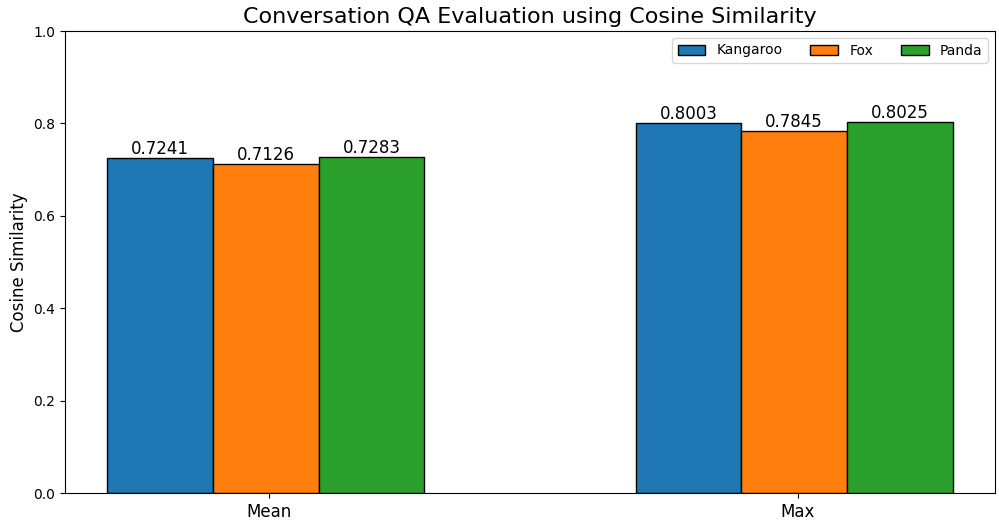
- $\mathbf{A}$ is the embedding vector of a retrieved document
- $\mathbf{B}$ is the embedding vector of a generated answer

1.5 Evaluation

To analyze the model's performance, two key statistical measures are computed for each answer:

- *Mean Cosine Similarity:* The average cosine similarity between the generated answer and the retrieved documents. This provides an overall measure of how well the answer matches the relevant context in general.
- *Max Cosine Similarity:* The highest cosine similarity between the generated answer and any of the retrieved documents. This indicates the best match between the answer and the context.

Figure 1: Conversation QA Evaluation using Cosine Similarity



1.6 Conclusion

The **Panda** model shows the best results with 0.72 mean and 0.80 max cosine similarity for 3 retrieved documents.

2 Task - Arbeitszeugnis Generation

2.1 Introduction

Large language models (LLMs) have shown tremendous potential in the realm of creative text generation, offering an efficient solution to tasks that typically require a lot of manual labor and time-consuming effort. Whether it's crafting stories, generating product descriptions, or even producing marketing copy, LLMs can help reduce the effort and time spent on these creative tasks while adhering to specific constraints and reproducing patterns on new inputs. Their ability to generate diverse and high-quality content at scale allows businesses and creators to focus on higher-level strategic work, while automating the repetitive aspects of content generation.

In this task, LLMs are used to generate job references for employees leaving a company by taking inputs such as job title, duration, and career level. By leveraging these inputs, the models produce personalized, professional references quickly, saving time and effort. This eliminates the need for HR departments to manually craft individualized references, which can be time-intensive, especially when dealing with a high volume of employees. The generated references adhere to safety guidelines, avoiding bias and ensuring fairness. Through prompt engineering, and using techniques like few-shot prompting, LLMs can be employed to ensure the text is harmless, accurate, and consistent with company standards, making them a valuable tool for HR departments. Furthermore, LLMs can reflect the unique culture, tone, and values of the company, providing customized references that maintain a professional and positive tone while being compliant with legal and ethical standards.

2.2 Data

The HR department shared three examples of job reference, which were included in the prompt, along with a set of 10 new inputs for generating job reference.

2.3 Methodology

2.3.1 Inputs

The large language model (LLM) is provided with various inputs, such as job title, duration, career level, and other relevant details, which serve as the foundation for generating a job reference. These inputs guide the model in producing a tailored and contextually appropriate reference that aligns with the given parameters.

2.3.2 Ethical Considerations

The LLM is given guidelines that aim to promote inclusive, neutral, and respectful language. They are encouraged to use general role descriptions instead of personal names, avoid gender-specific terms, and eliminate personal pronouns to ensure neutrality. The guidelines also emphasize simple sentence structures, respectful language, and avoiding formal punctuation like colons in bullet points. The goal is to create content that is inclusive and accessible to everyone.

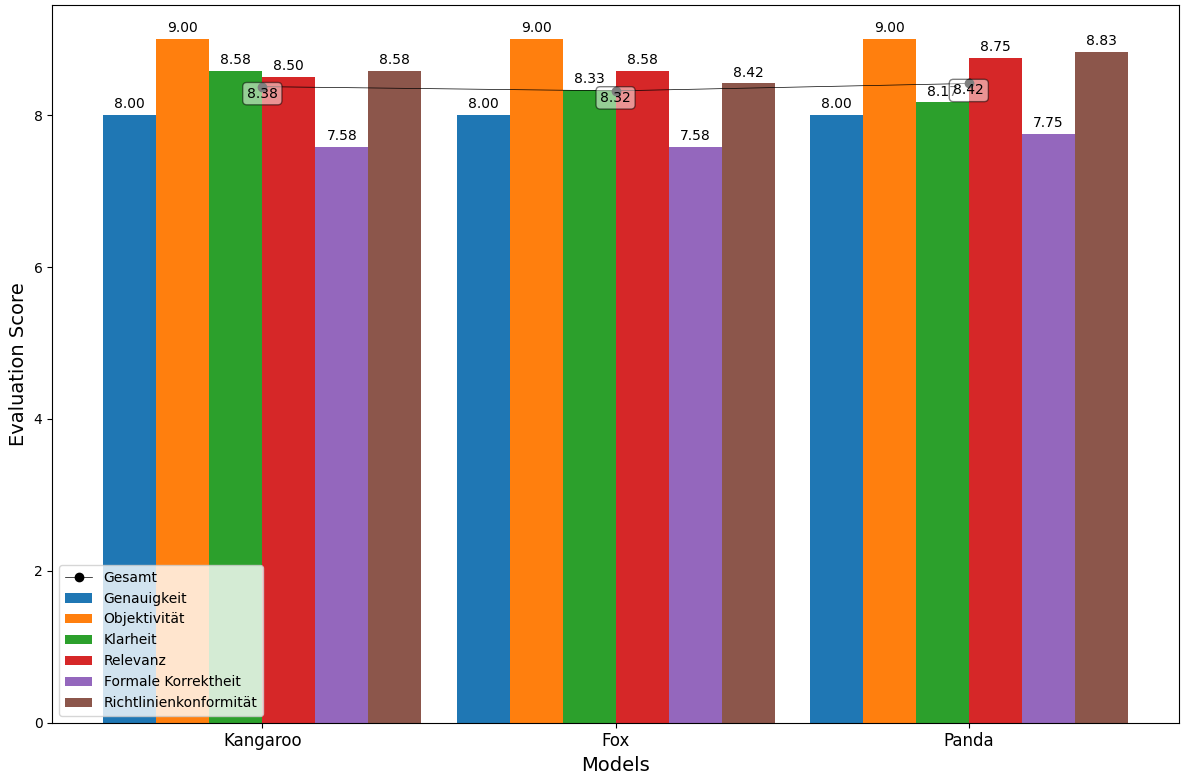
2.3.3 Few-shot Prompting

To enhance the LLM's ability to generate job references that align with the given inputs, a technique known as few-shot prompting is employed. This involves providing the model with a series of example inputs paired with their corresponding job references. By learning from these examples, the model gains the ability to generalize and produce similar, contextually relevant outputs for new, previously unseen input sets. This approach ensures that the model can handle a wide range of job reference scenarios while maintaining consistency and relevance in its output.

2.4 Evaluation

A separate and distinct large language model (LLM) was utilized to evaluate the job references generated by the initial models. The evaluation process focused on a comprehensive set of criteria, including accuracy, objectivity, clarity, relevance, formal correctness, and adherence to established guidelines. This secondary model assessed the content produced by the first LLM to ensure that it met the desired standards in each of these areas. Specifically, the evaluation aimed to verify whether the generated references were factually accurate, free from bias, easy to understand, contextually appropriate, linguistically formal, and in strict compliance with the ethical and stylistic guidelines provided. This multi-layered approach to assessment ensured that the job references maintained high quality and aligned with the set expectations.

Figure 2: Arbeitszeugnis Evaluation



2.5 Conclusion

The **Panda** model demonstrates the most impressive overall performance, achieving a noteworthy score of 8.42. It excels particularly in the areas of relevance, formal correctness, and strict adherence to the established guidelines.

2.6 Challenges

The entire task was conducted entirely in the German language. This encompassed not only the inputs provided to the model but also the outputs generated, as well as the prompt used to guide the task.

3 Task - VBA Macro Code Analysis

3.1 Introduction

Large language models have become increasingly important in improving the way developers interact with and manage code. By leveraging their advanced capabilities, LLMs help developers better understand, summarize, and analyze code, making it easier to work with even the most complex codebases. These models excel at parsing and explaining code, providing clear, detailed descriptions of its structure, logic, and purpose. This allows developers to grasp the functionality of existing code, even when they are not familiar with it.

LLMs can thoroughly analyze code to identify potential issues such as errors, inefficiencies, or security vulnerabilities. They can also provide valuable suggestions for improvement, enabling developers to optimize their code and ensure it runs more effectively. Another significant advantage of LLMs is their ability to automatically generate documentation, which enhances code maintainability by making it easier for others to understand the logic behind various sections of code. LLMs are also instrumental in debugging processes, as they can help pinpoint errors in the code and offer actionable recommendations for fixing them. Finally, LLMs serve as valuable educational tools, offering explanations of complex programming concepts. By using LLMs, developers can save time, minimize errors, and produce higher-quality, more efficient code.

In the case of an organization transitioning from Microsoft software to Google software, the process becomes even more complex due to the presence of thousands of Excel VBA macros that have been used to automate various tasks. These macros, integral to the organization's workflow, will eventually be deprecated as part of the shift away from Microsoft products. In this scenario, generative AI can play a crucial role as an initial tool in facilitating the migration process. As the organization moves away from Microsoft Excel, Gen AI can help by analyzing the existing macros, understanding their functions. This step is essential, as it helps the organization prepare for the eventual transition by providing insights into how the automation processes currently in place can be restructured to work effectively in the new environment. Ultimately, generative AI will assist developers by offering a streamlined approach to the migration process, ensuring that the transition from Microsoft to Google is as smooth and efficient as possible.

3.2 Data

The stakeholders provided the complete set of files from *five* VBA macro projects. These files included all necessary scripts, resources, and associated code required to understand and evaluate each project in detail. These projects ranged between 5000 to 50,000 lines of code.

3.3 Methodology

3.3.1 Understanding Requirements

A critical first step in the process was to clearly define the types of information that would be most beneficial for developers in understanding the existing VBA macros. The requirements were comprehensive and varied, ranging from technical details such as counting the number of classes, functions, and modules within the code, to higher-level tasks such as generating user stories, domain models, and other documentation that could assist developers in grasping the full scope and functionality of the macros. Identifying these needs helped to ensure that the generated outputs would align with the practical requirements of the developers, enabling them to effectively understand the existing code.

3.3.2 Parsing the Code

The VBA macros consisted of numerous files, each containing lines of code that needed to be thoroughly examined. The first step was to parse the contents of these files, extracting and structuring the code in a manner that could be easily processed by the LLM. Each file's content was read and transformed into a Python string, which could then be formatted into the input for the LLM. This step was vital for ensuring that the entire codebase was captured in a format that the model could analyze accurately and generate useful insights from.

3.3.3 Analyzing the Code

The model was tasked with producing a detailed report that could serve as a reference for developers. This report needed to break down the functionality and contents of the macro code in a way that was easily understandable, helping developers identify the logic behind the code, its components, and how they interrelate. The goal was to ensure that the report provided clear insights into the structure and behavior of the macros, offering a comprehensive understanding that would be valuable during the process of reviewing and transitioning the macros.

3.3.4 Save Report

The LLM generated its analysis and report in markdown format, which proved to be a practical and versatile choice for saving and sharing the output. Markdown files are easily convertible to other formats such as HTML or PDF, making them ideal for documentation purposes. To ensure that the final report was well-organized and accessible, explicit instructions were provided to the LLM on how to structure the content. These instructions aimed to guarantee that the report was not only thorough and informative but also easy to read, with clear headings, sections, and formatting that would allow developers to quickly find the information they needed.

3.4 Evaluation

To ensure the final report was comprehensive and well-structured, fallback strategies were put in place to address any potential issues with the LLM's responses. If the generated reports were found to be incomplete or inaccurate, they were re-processed, allowing for corrections and refinements until the desired level of completeness and accuracy was achieved. Given the importance of delivering a detailed and coherent report, this iterative process ensured that the quality of the output was consistently high.

Considering the length of the reports, which ranged from 20 to 30 pages, and the fact that these reports were intended to be practical and accessible for developers, a human evaluation approach was adopted. Since the reports were dense with technical details and required a nuanced understanding of the code, human evaluators were best suited to assess the quality, relevance, and clarity of the generated content. These evaluations allowed for an in-depth review of how well the reports met the intended objectives and whether they provided developers with the necessary information to understand the existing macro code. The reports were generated using three different large language models. The model names were kept anonymous to prevent any bias in the evaluation process. These reports were then shared with stakeholders for further review and feedback, allowing the development team to compare and contrast the outputs and choose the most useful and effective approach.

Additionally, an analysis of the time required to generate a complete report for a single macro was conducted. This analysis helped assess the efficiency of the process and provided valuable insights into the time and resources needed to produce high-quality reports for the entire macro codebase.

Figure 3: Time taken - Individual Macro

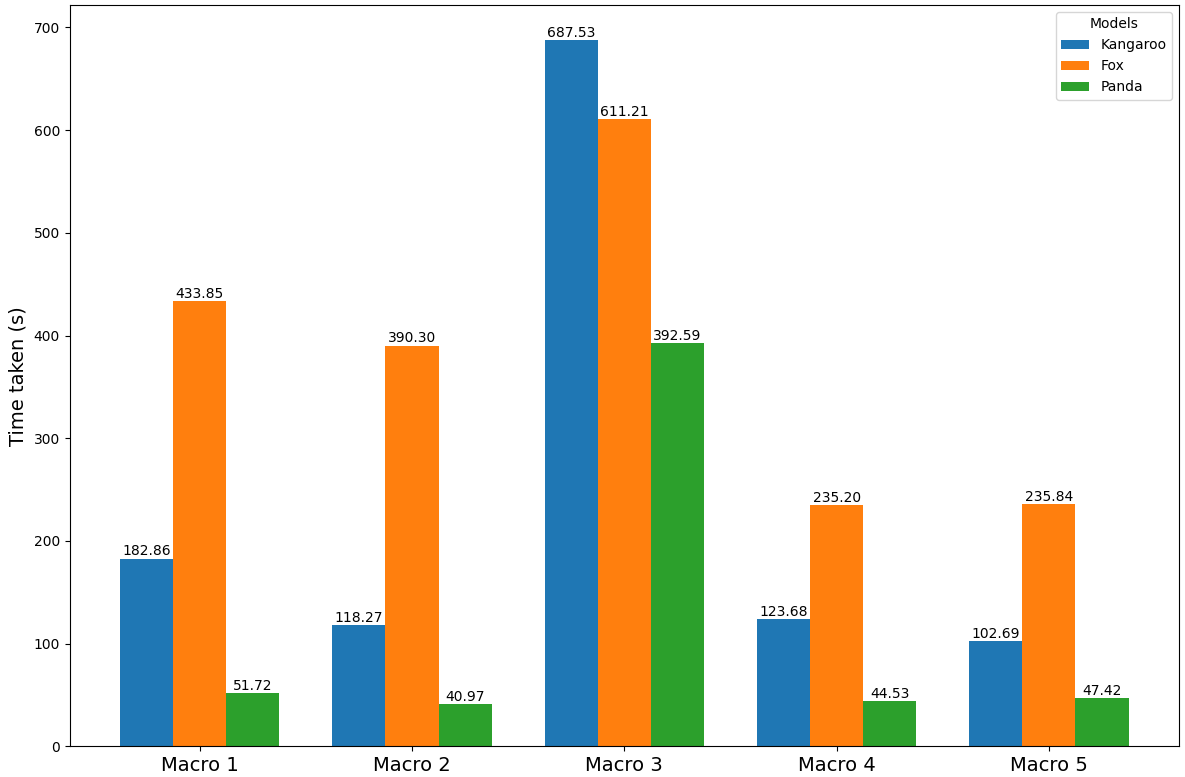
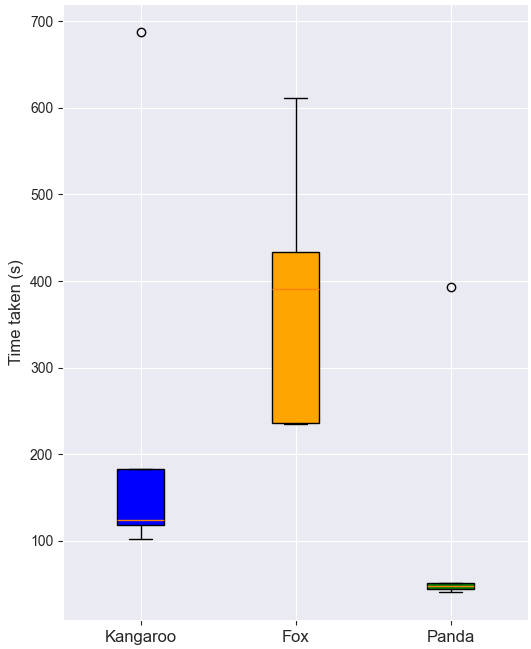


Figure 4: Time taken - Distribution



3.5 Conclusion

The **Panda** model is the most efficient at generating code analysis reports, typically producing a report for most macros in under a minute.

3.6 Challenges

The lack of domain knowledge in VBA, made it difficult to fully evaluate the content and accuracy of the generated reports. Without a deep understanding of the code, it was a challenge to ensure the generated reports met the expectations and requirements of the stakeholders.

The codebase could be quite large, making it critical to choose an LLM with a large context token limit. Having a model with a sufficient token limit was crucial to ensure that the entire codebase could be processed effectively in a single pass.

Due to the output token limit of the LLM, the generated reports needed to be broken down into multiple API calls. It required careful management to ensure the entire report was accurately generated and pieced together from the various API responses. Balancing the need for detailed, thorough reports with the constraints of token limits required a thoughtful approach to structuring the data and ensuring that nothing was overlooked.

I took on complete responsibility for managing the task, which included understanding the requirements from the stakeholders, maintaining constant communication, and providing regular updates throughout the process. This also involved scheduling meetings, taking detailed notes, and sharing the final reports with stakeholders for review and feedback. Managing these responsibilities required clear communication to ensure the process ran smoothly and that all expectations were met.

4 Tools and Technology

1. Python
2. Jupyter Notebook
3. Langchain
4. Vertex AI SDK Python
5. Matplotlib
6. Numpy
7. Prompt Engineering
8. Zero-shot and few-shot prompting
9. API Integration

5 Lessons Learned

1. **Accountability:** One of the most significant lessons gained during this internship was the importance of taking full ownership of assigned tasks. This involved not only executing the work independently but also being accountable for its quality, accuracy, and completeness. Ensuring high standards in deliverables became a core focus, reinforcing a commitment to excellence throughout the project.

2. **Time Management:** Meeting deadlines consistently was crucial to the success of the projects. This experience emphasized the value of disciplined time management, strategic planning, and prioritization. I learned to break down complex tasks into manageable milestones, allocate time effectively, and maintain a steady pace to ensure timely completion of all deliverables.

3. **Clear Communication:** Efficient communication proved essential for collaboration, especially in a fast-paced and dynamic work environment. I developed the ability to convey ideas clearly, share updates proactively, and provide or seek feedback constructively. This fostered smoother coordination, minimized misunderstandings, and contributed to a more productive workflow.

4. **Presentation Skills:** Throughout the course of the internship, I had multiple opportunities to present ideas, progress, and outcomes to diverse audiences. This helped me structure content logically, and use visual aids effectively. I also became more confident in tailoring presentations to suit the audience's level of understanding and interest