Optimizing Tool-Based Query Solving Using Fine-Tuned Large Language Models

Rahul Vimalkanth

Abstract

This paper presents a novel approach for leveraging fine-tuned Large Language Models (LLMs) to automate tool-based query solving. As part of the InterIIT Hackathon 2024, the objective was to utilize Groq's LLaMA3 model to solve complex user queries by generating and refining a sequence of tool calls. By employing a two-stage LLM architecture, this system effectively integrates predefined tools to execute tasks such as querying databases, summarizing data, and resolving work items. The results, based on a rigorous evaluation using ROUGE metrics, demonstrate high accuracy, low latency, and scalability in handling real-world scenarios. Our study explores the effectiveness of LLM prompting, limitations in tool selection, and potential improvements in future iterations.

1. Introduction

Large Language Models (LLMs) have seen significant advancements in recent years, particularly in their ability to handle diverse tasks across multiple domains. However, their integration with structured tool-based systems for automating real-world queries is an evolving challenge. In this paper, we explore the use of LLMs to solve predefined queries by leveraging a set of external tools, as part of the InterIIT Hackathon 2024.

The task requires the generation of accurate tool calls based on user queries, ensuring adherence to a specific JSON schema. To address this, we utilize Groq's fine-tuned *LLaMA3* model, a high-performing open-sourced model designed for tool use. Groq's API, which is available at no cost, supports seamless integration of LLMs with tools, and currently ranks highly on the Berkeley Function Calling Leaderboard.

Our system employs a two-stage LLM architecture to optimize the process of tool selection and argument filling, ultimately solving queries efficiently and accurately. This design, coupled with a comprehensive evaluation methodology, provides insights into LLM capabilities in real-world task automation.

2. Related Work

Previous studies have explored the use of LLMs for function calling, focusing on integrating natural language understanding with external systems. Recent advancements in fine-tuning models such as LLaMA3 for task-specific purposes have demonstrated improved tool utilization, as seen in benchmarks like the Berkeley Function Calling Leaderboard.

Our approach builds on this foundation, investigating the specific problem of tool-based query solving in a constrained environment. While approaches such as Retrieval-Augmented Generation (RAG) have shown promise, limitations in handling predefined tools and schema adherence remain open research challenges.

3. System Architecture

3.1. Tool-Based Interaction Framework

The interaction framework is centered around a predefined set of tools provided for query resolution. These tools include functions such as `works_list`, `summarize_objects`, and `search_object_by_name`, each designed to handle a specific task within the system. Each tool is equipped with a list of arguments, and the correct selection and invocation of these tools determine the success of the query resolution.

To ensure efficient tool utilization, we adopted a pipeline that leverages two distinct LLMs:

- **MODEL1**: A first-pass model that generates an initial sequence of tool calls based solely on the user query. It receives no additional context from prior examples.
- **MODEL2**: A second-pass model that refines the output of MODEL1 by utilizing examples, improving the accuracy of tool selection and argument completion.

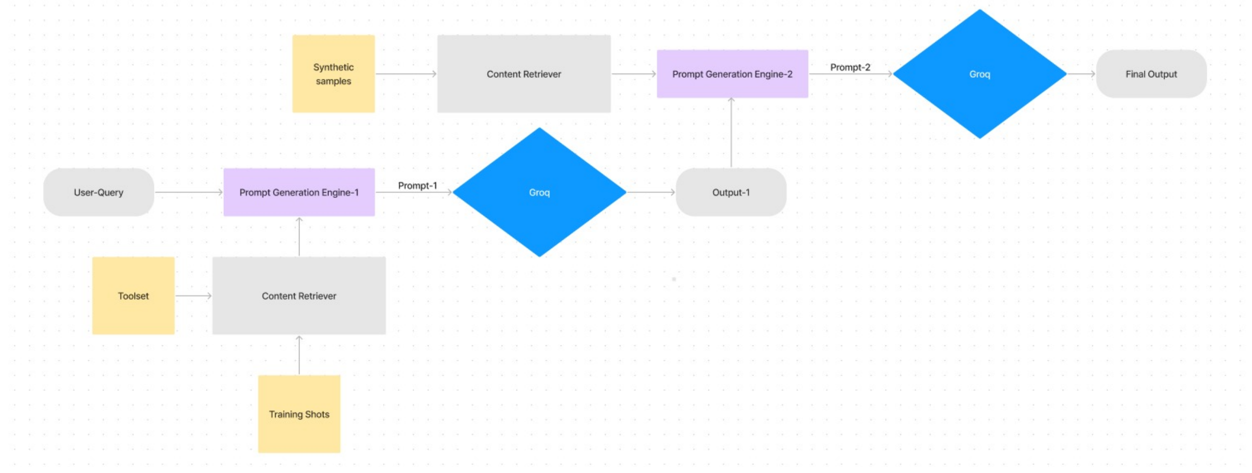

Figure 1: Flowchart of the Model Architecture

3.2. Model Selection and Justification

We experimented with several models, including Mistral, Claude, and the xLAM series, which were ranked highly in the Berkeley Function Calling Leaderboard. However, the LLaMA3-Groq-70B-8192-tool-use-preview model outperformed the others in terms of both accuracy and latency. Groq's model is specifically fine-tuned for tool calling, making it well-suited for the constrained task environment of the hackathon. Moreover, the complexity of the task made fine-tuning on the provided dataset unnecessary due to its size, which was smaller than the context window of the model. Instead, prompting was sufficient.

Initial tests showed that MODEL1, with access to the tools and two example queries, demonstrated low latency and high accuracy in understanding and generating tool calls in the required JSON format. However, it struggled with more complex queries that required the use of multiple tools in sequence.

4. Methodology

4.1. Prompting Strategy

Given the limitations of the provided dataset, we implemented a refined prompting strategy. By feeding MODEL1 with only tools and basic examples, it was able to generalize well. MODEL2, which had access to additional synthetic data generated from the problem statement, refined the output of MODEL1, ensuring the final result met the task's requirements.

To address the complexity of tool sequences, I generated 15+ synthetic datasets, where each query required specific tool usage, often referencing prior tools' outputs using `$$PREV[i]`. This approach enabled the system to handle diverse, complex scenarios involving multiple tool calls, significantly enhancing its real-world applicability.

4.2. Challenges with Alternative Models

While LLaMA3 performed well, other models presented challenges. For example, the xLAM model required a specific format for tool inputs, which proved difficult to implement due to a lack of comprehensive documentation. After testing models such as Mistral and Claude, I determined that LLaMA3-Groq was the most suitable for this task.

4.3. Pipeline Workflow

The workflow consists of the following stages:

1. **Query Input**: The user inputs a query, such as "List all high-severity tickets coming from Slack for customer Cust123."
2. **MODEL1 Execution**: MODEL1 generates a sequence of tool calls based on the query, selecting appropriate tools and arguments.
3. **MODEL2 Refinement**: MODEL2 refines MODEL1's output by incorporating synthetic data and examples, ensuring accurate tool calls and argument usage.
4. **Tool Execution**: The final set of tool calls is executed in sequence to produce the desired output.

This two-model approach minimizes the risk of overwhelming the model with large prompts while ensuring the use of relevant tools in the correct sequence.

5. Evaluation

5.1. Performance Metrics

To assess the system's performance, we used ROUGE scores to compare the model's output against ground-truth responses. ROUGE-1 and ROUGE-L metrics were used to measure n-gram precision and recall, as well as the longest matching subsequence between the model output and the expected result.

The system consistently achieved ROUGE scores between 60 and 80, demonstrating strong performance across both simple and complex queries. The average latency per query was approximately 6 seconds, although this increased to over 10 seconds for particularly complex queries.

5.2. Limitations and Hallucination

Although the system performed well, hallucination remained a challenge. On occasion, the model returned excessive tool calls with empty arguments, or assumed argument values, leading to erroneous outputs. These issues were mitigated in part by the use of two models, but further refinement is necessary.

5.3. Potential for Retrieval-Augmented Generation (RAG)

In a later phase, we explored the potential for implementing Retrieval-Augmented Generation (RAG) by storing example queries in a VectorDB using Langchain and ChromaDB. This would allow MODEL2 to retrieve relevant examples based on semantic similarity, providing a more structured approach to handling complex queries. While promising, this was not fully implemented due to time constraints.

6. Conclusion

This study demonstrates the effectiveness of using fine-tuned LLMs, particularly LLaMA3, for solving tool-based queries in structured environments. The two-model approach improved the system's ability to handle complex scenarios without overwhelming the model. However, challenges such as hallucination and tool argument errors highlight areas for future work. The system's ability to scale with minimal fine-tuning makes it a promising solution for real-world applications requiring structured query resolution.

7. Future Work

To further enhance the system's capabilities, future efforts will focus on:

- Implementing a complete RAG system for more efficient example retrieval.
- Reducing latency for complex queries through model optimization.
- Addressing hallucination by refining the model's understanding of tool argument dependencies.