Fine-Tuned Large Language Models into Traditional Back-End Web Architectures

Bowen Li

bowen.li@rmit.edu.au

Royal Melbourne Institute of Technology

Melbourne, Victoria, Australia

Chuan Zhang

chuan.z@hotmail.com

SkywardAI

Abstract

Integrating Large Language Models (LLMs) into traditional back-end systems can significantly reduce development overhead and enhance flexibility. This paper presents a novel approach using a fine-tuned LLama3 model as a modular back-end component capable of processing JSON-formatted inputs and outputs. We developed a specialized dataset through advanced prompt engineering with the Phi-3.5 model and fine-tuned LLama3 using Quantized Low-Rank Adaptation (QLoRA) on a single NVIDIA T4 GPU. An API layer was designed to facilitate seamless communication between clients and the LLM, effectively replacing conventional application logic. Our fine-tuned model achieved an average accuracy of 76.5% and a response time of 3.56 seconds across 100 test cases, demonstrating its effectiveness in handling back-end tasks. This work underscores the potential of LLMs to transform AI-driven back-end architectures, offering scalable and efficient solutions for modern web services. The source code is publicly available at <https://github.com/SkywardAI/chimera>.

Introduction

A back-end web server operates as the hidden backbone of a website, managing logic and data processing to support user-facing interactions. It is responsible for receiving requests from clients’ web browsers, processing them, interacting with databases to retrieve or store information, and delivering appropriate responses to the user’s browser. The core components of a back-end server include three key parts: the server (hardware), the application, and the database Belloum et al. (2002). The database maintains the system state, while the application handle incoming requests, processes data, and generate responses. Both databases and applications operate on physical machine or cloud-based environments. Among these components, the application often requires extensive development resources to implement and maintain.

Recent advances in artificial intelligence (AI) have introduced Large Language Models (LLMs), which excel at processing, understanding, and generating human-like text. However, this language-centric capability can extend beyond human languages. For instance, JSON-formatted messages, a common communication format in web services, can be treated as a structured language. By leveraging this perspective, LLMs can emulate the logic and behavior of traditional back-end applications.

To tap into this potential, LLMs must be tailored to the specific needs and workflows of a given back-end. Fine-tuning—a machine learning technique that further trains a pre-trained model on a domain-specific dataset—offers a promising solution. Through fine-tuning, LLMs can evolve from general-purpose language generators into specialized components that accurately process input, apply custom logic, and return responses in prescribed formats.

The contribution of this paper is to demonstrate how an LLM can be fine-tuned to function as a modular component within a traditional back-end architecture. Specifically, we show how to adapt a pre-trained LLM to produce responses in a predefined, structured format—akin to a custom back-end application. This approach highlights a practical pathway for integrating LLMs into conventional web back-end servers, potentially reducing development overhead and streamlining the implementation of specialized logic.

Related Work

In this section, we review related work on integrating LLMs into traditional web services.

Recent advances in large language models (LLMs) have prompted a surge in research exploring their integration into various back-end and service-oriented architectures. For instance, RestGPT[7] introduces a framework allowing LLMs to interact directly with RESTful APIs, effectively enabling them to perform complex tasks by decomposing instructions and selecting appropriate API endpoints. This approach demonstrates how LLMs can serve as dynamic components that replace certain traditional back-end functions. Similarly, Large Search Model[8] proposes a unified framework that employs LLMs to handle traditionally distinct aspects of the search pipeline, such as query interpretation, retrieval, and ranking. By formulating these operations as autoregressive text generation problems, LLMs streamline the search process and potentially reduce the complexity of conventional, modularized search stacks.

Another direction, exemplified by the “Sahaay” system[5] and the efforts in e-government services[6], shows how LLMs can be harnessed to replace human-driven or fixed-rule components in more specialized applications. In the case of Sahaay, LLMs integrate with customer service platforms, automating tasks like query resolution and response generation, thereby reducing the need for dedicated human agents or individually tailored modules. Meanwhile, in the e-government domain, researchers have explored Retrieval-Augmented Generation architectures to support public services, illustrating that LLMs can facilitate and improve processes previously reliant on rigid, manually-coded logic.

While existing studies highlight the potential of LLMs in back-end tasks by using pre-trained models or generic prompts, our work fine-tunes LLMs to function as modular back-end components with application-specific responses. This customization bridges the gap between flexible language processing and the rigid requirements of traditional systems. Unlike approaches that rely on extensive prompting or unchanged models, our method systematically refines LLMs to accurately implement domain logic and response structures tailored to the target application.

Our contribution lies in demonstrating how fine-tuning can transform an LLM into a dedicated, conventional back-end service. By adapting the model to a specific back-end role and ensuring it generates domain-specific output formats, we offer a practical, efficient pathway for integrating these models as drop-in replacements or supplementary components. In doing so, our approach reduces the complexity and development overhead of traditional server-side logic, ultimately paving the way for more flexible, AI-driven back-end architectures.

Methodology

This section outlines the methodology for integrating a fine-tuned Large Language Model (LLM) into a traditional back-end system, encompassing dataset creation, API layer design and implementation, deployment of the LLM as a service, and the overall system workflow.

Dataset Creation

To effectively fine-tune the Large Language Model (LLM) for handling JSON-formatted inputs and outputs, we developed a specialized dataset[2] tailored to the requirements of our back-end system. The dataset creation process is pivotal, as it directly influences the model’s ability to accurately parse, process, and generate structured responses. This subsection delineates the steps undertaken to create the dataset, including its composition, generation methodology, and the computational resources utilized.

Dataset Composition

The dataset comprises 5,000 entries, each structured to facilitate the training of the LLM in handling diverse types of inputs and generating corresponding JSON-formatted outputs. Each entry in the dataset consists of four columns:

* **Output String:** The desired JSON-formatted response that the model should generate.
* **Structured Input String:** A JSON-formatted request that the model needs to process.
* **Direct Input String:** A direct, non-structured query related to the structured input.
* **Conversational Input String:** A conversationally phrased version of the direct input query.

| *Table 1: Different format prompt for the same structured output* | |
| --- | --- |
| **Input** | **Output** |
| "{“A”:74, “op”:‘\*’’, “B”:70}" | "{“result”: “5180”}" |
| What is the result of multiplying 74 by 70? | "{“result”: “5180”}" |
| Calculate the result when you multiply 74 by 70? | "{“result”: “5180”}" |

Dataset Generation Methodology

The dataset was generated using the Phi-3.5 model[1] through meticulous prompt engineering. This approach involved designing specific prompts that instruct Phi-3.5 to produce the required structured and unstructured inputs alongside their corresponding outputs. The process ensured that the dataset encapsulated a wide variety of request types and response scenarios, enhancing the LLM’s ability to generalize across different contexts.

#### Prompt Engineering

Prompt engineering was employed to guide Phi-3.5 in generating high-quality, diverse data entries.

PROMPT\_TEMPLATE = [  
 {  
 "role": "system",  
 "content": "You are an AI assistant that converts structured prompts into {conversion\_type} questions and no need to answer it"  
 },  
 {  
 "role": "user",  
 "content": (  
 "Convert the following structured prompt into a {conversion\_type} question:\n"  
 "Structured Prompt: {structured\_input}\n"  
 )  
 }  
]

#### Data Diversity and Coverage

To address the fundamental aspects of back-end API interactions, our dataset is focused on basic arithmetic operations, specifically addition (+), subtraction (-), multiplication (\*), and division (/). The dataset comprises fewer than 100 entries, each representing a distinct arithmetic computation. While the current dataset is limited in scope, it serves as a critical initial step towards training the LLM for more complex operations and diverse business logic implementations in future iterations.

Customized Dataset Development and Reproducible Workflow

Our primary contribution is the development of a specialized dataset tailored for training Large Language Models (LLMs) to process and generate structured JSON inputs and outputs. Utilizing the Phi-3.5 model and advanced prompt engineering techniques, we created a dataset that enables LLMs to accurately interpret diverse input formats and produce corresponding JSON-formatted responses. Additionally, we established a reproducible workflow that allows researchers and practitioners to customize and generate similar datasets tailored to their specific structured output requirements. This dual contribution not only enhances the integration of LLMs into traditional back-end systems by ensuring precise data handling but also provides a scalable framework for future dataset creation efforts, fostering consistency and reliability in training LLMs for structured data tasks.

Supervised Fine-Tuning the Large Language Model

We performed supervised fine-tuning of the LLama3 model[4] using our custom dataset, employing the QLoRA (Quantized Low-Rank Adaptation)[3] to efficiently adapt the model with limited computational resources. This fine-tuning process was executed on a single Nvidia-T4 GPU, enabling the model to learn to accurately interpret and generate structured JSON outputs based on the diverse inputs provided in the dataset. By leveraging QLoRA, we optimized memory usage and accelerated training times without compromising the model’s performance. The resulting fine-tuned LLama3 model is now capable of reliably handling both machine-readable and human-readable requests within our back-end system, enhancing its overall functionality and responsiveness.

API Layer Design and Implementation

The API Layer is responsible for managing the communication between the client and the server. It formats and interprets RESTful requests and responses, handling tasks such as routing, data validation, and serialization. This layer ensures that requests from clients conform to the expected structure and that responses from the server are appropriately formatted for delivery. By isolating this functionality into a dedicated layer, we decouple the communication mechanics from the back-end’s core logic, allowing the LLM to focus exclusively on higher-order tasks.

The API Layer also interfaces with the database. However, unlike traditional back-ends, it doesn’t directly execute database operations. Instead, it forwards database requests generated by the Core Logic Unit to the database service, treating it similarly to a front-end client.

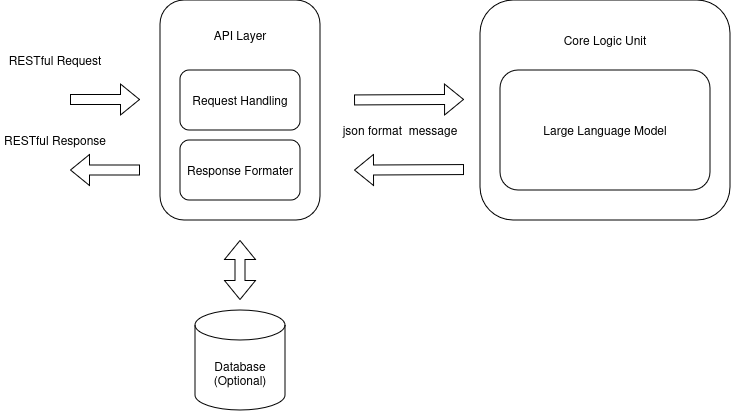
Integration of the Fine-tuned LLM as a Service

The Core Logic Unit contains the bulk of the back-end’s application logic. Traditionally, this unit handles tasks such as business logic, database interactions, and processing rules specific to the application. In our design, this unit is replaced by an LLM, which is fine-tuned to emulate and execute the core logic of the application. The LLM processes structured inputs received from the API Layer, performs the necessary computations or decision-making processes, and generates structured outputs for the API Layer to format and deliver.

System Architecture

The system architecture is shown in Figure1.

* Request Handling: A RESTful request from the client is received by the API Layer, which validates and formats the request into a structured format, such as JSON.
* Core Logic Execution: The structured request is passed to the LLM in the Core Logic Unit, which processes the input according to the application’s logic and generates an appropriate response.
* Response Formatting: The API Layer formats the LLM’s output into a RESTful response and sends it back to the client.



*Figure 1 System Architecture*

This modular design ensures flexibility and scalability, allowing the API Layer to handle evolving communication protocols while the LLM adapts to increasingly complex back-end logic through iterative fine-tuning. The separation of concerns also facilitates maintenance and testing, as each layer can be developed and validated independently. Our methodology demonstrates how LLMs can effectively replace traditional application logic in back-end systems, paving the way for AI-driven web back-end architectures.

Result

This section presents the experimental results demonstrating the performance of the fine-tuned LLama3 model integrated into the back-end system.

Performance Metrics from the Model Fine-Tuning Process

We summarized the fine-tuning results using WandB. Results are shown in Table 2.

| *Table 2 Detailed Evaluation and Training Metrics for Fine-Tuned Model* | |
| --- | --- |
| **Metric** | **Value** |
| eval/loss | 0.40704 |
| eval/runtime | 26.0878 |
| eval/samples\_per\_second | 3.833 |
| eval/steps\_per\_second | 3.833 |
| total\_flos |  |
| train/epoch | 1 |
| train/global\_step | 450 |
| train/grad\_norm | 1.03545 |
| train/learning\_rate | 0 |
| train/loss | 0.4144 |
| train\_runtime | 670.3678 |
| train\_samples\_per\_second | 1.343 |
| train\_steps\_per\_second | 0.671 |

API Layer Accuracy and Response Time

We use the fine-tuned model as the Core Logic and test result accuracy and timing. The total number of test case is 100. Results are shown in Table 3.

| *Tabel 3 Performance Metrics of the API Layer* | |
| --- | --- |
| Average accuracy | 0.765 |
| Average response time | 3.56 s |

Limitations and Future Work

Limitations

Despite the successful integration of the fine-tuned Large Language Model (LLM) into our back-end system, several limitations must be acknowledged:

* **Dataset Size and Computing Resources:** The relatively small size of our dataset, coupled with limited computing resources, constrains the accuracy and generalizability of the trained model. A larger and more diverse dataset, alongside enhanced computational capabilities, could facilitate more nuanced learning and improve the model’s performance across a broader range of tasks.
* **Response Time:** While the LLM-based service demonstrates stability across all tasks, its response time is noticeably slower compared to traditional back-end services. This latency may impact real-time applications where swift responses are critical. Although the system maintains consistent performance, optimizing the model architecture and deployment strategies is necessary to achieve response times comparable to conventional services.

Future Work

To overcome current limitations and enhance the system, future work will focus on:

* **Expanding the Dataset and Upgrading Computational Resources:** Increasing the dataset size and diversity will improve the LLM’s accuracy and adaptability. Additionally, enhancing computing infrastructure will allow for more extensive training and better performance.
* **Optimizing Model Efficiency and Reducing Latency:** Implementing optimization techniques like quantization and pruning can lower computational demands and speed up response times. Exploring specialized hardware and alternative deployment frameworks will further enhance efficiency without compromising stability.

These improvements will address existing limitations and boost the system’s effectiveness in real-world applications.

Conclusions

This study presents a specialized dataset for training Large Language Models (LLMs) to handle structured JSON inputs and outputs, addressing a significant research gap. Utilizing the Phi3.5 model and advanced prompt engineering techniques, we created a robust dataset that enables LLama3 to accurately interpret and generate JSON-formatted responses. Additionally, we established a reproducible workflow, allowing other researchers and practitioners to develop customized datasets for integrating LLMs into traditional back-end systems. Despite limitations in dataset size and computational resources, our approach demonstrates the potential of fine-tuned LLMs to enhance the functionality and responsiveness of back-end services. This work lays the foundation for future advancements in training methodologies and dataset expansion, paving the way for more efficient and accurate integrations of LLMs in diverse operational environments.

References

1. Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H. H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H. S., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chopra, P., Del Giorno, A., et al. (2024). *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. ArXiv abs/2404.14219. Available at: <https://api.semanticscholar.org/CorpusID:269293048>.
2. Aisuko. (2024). *Diverse Calculation*. Available at: <https://doi.org/10.57967/hf/3724>. Accessed: 2024-12-09.
3. Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs*. arXiv:2305.14314 [cs.LG]. Available at: <https://arxiv.org/abs/2305.14314>.
4. AI @ Meta Llama Team. (2024). *The Llama 3 Herd of Models*. arXiv:2407.21783 [cs.AI]. Available at: <https://arxiv.org/abs/2407.21783>.
5. Pandya, K., & Holia, M. (2023). *Automating Customer Service using LangChain: Building Custom Open-Source GPT Chatbot for Organizations*. arXiv:2310.05421 [cs.CL]. Available at: <https://arxiv.org/abs/2310.05421>.
6. Papageorgiou, G., Sarlis, V., Maragoudakis, M., & Tjortjis, C. (2024). *Enhancing E-Government Services through State-of-the-Art, Modular, and Reproducible Architecture over Large Language Models*. *Applied Sciences*, 14(18), 8259. <https://doi.org/10.3390/app14188259>.
7. Song, Y., Xiong, W., Zhu, D., Wu, W., Qian, H., Song, M., Huang, H., Li, C., Wang, K., Yao, R., Tian, Y., & Li, S. (2023). *RestGPT: Connecting Large Language Models with Real-World RESTful APIs*. arXiv:2306.06624 [cs.CL]. Available at: <https://arxiv.org/abs/2306.06624>.
8. Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., & Wei, F. (2023). *Large Search Model: Redefining Search Stack in the Era of LLMs*. arXiv:2310.14587 [cs.IR]. Available at: <https://arxiv.org/abs/2310.14587>.