RESTful-Llama: Connecting User Queries to RESTful APIs

Han Xu[1], **Ruining Zhao**[1], **Jindong Wang**[2], **Haipeng Chen**[2]
[1]University of Illinois at Urbana-Champaign
[2]William & Mary
{hanxu8, ruining9}@illinois.edu, {jwang80, hchen23}@wm.edu

Abstract

Recent advancements in Large Language Models (LLMs) have showcased exceptional performance in zero-shot learning and reasoning tasks. However, integrating these models with external tools - a crucial need for real-world applications - remains a significant challenge. We propose RESTful-Llama, a novel framework designed to enable Llama 3.1 to transform natural language instructions into effective RESTful API calls. To enhance the fine-tuning process, we introduce DOC_Mine, a method to generate fine-tuning datasets from public API documentation. RESTful-Llama distinguishes itself by enabling open-source LLMs to efficiently interact with and adapt to any REST API system. Experiments demonstrate a 31.9% improvement in robustness and a 2.33x increase in efficiency compared to existing methods.

1 Introduction

Large language models (LLMs) have made significant strides in natural language processing (NLP) and various interdisciplinary domains in recent years. They exhibit the ability to engage in human-like conversations and demonstrate the potential to integrate with external tools, such as search engines and productivity software (Shen, 2024)—an essential feature for real-world applications. Given the widespread use of these tools in daily activities, such integration is critical to meeting end-user needs and enhancing their interaction with technology.

Building on this potential, a promising area of research seeks to incorporate LLMs with multi-modal tools. Intelligent planners like Visual ChatGPT (Wu et al., 2023) and HuggingGPT (Shen et al., 2023) utilize pre-defined templates to generate instructions executable by various foundation models. While these strategies have shown impressive results, they are typically confined to a limited selection of specially designed tools or models, making them difficult to adapt or extend to other systems. Moreover, the reliance on proprietary LLMs, such as ChatGPT (OpenAI, 2024a), raises concerns in the industry about potential data breaches.

Rather than focusing exclusively on a limited set of external tools, recent research efforts focus on improving LLM generalization across a wider range of tasks. For example, ReAct (Yao et al., 2023) enables LLMs to interact with external environments such as ALFWorld (Shridhar et al., 2021) and Wikipedia to tackle general tasks. Concurrently, Gorilla (Patil et al., 2023) facilitates machine learning-related API calls through platforms like TorchHub (PyTorch, 2023). Despite their broader scope, these approaches often face limitations in task resolution success rates, which impedes their practical deployment in real-world scenarios. On the other hand, these methods struggle with scenarios where flexibility across diverse API systems is required, such as when integrating LLMs with dynamic, real-world API systems that evolve over time.

To overcome the limitations of previous methods, this study introduces a novel method, **RESTful-Llama**, which empowers open-source LLMs to translate user natural language queries into effective RESTful API calls for real-world applications. In summary, the main contributions of the study are as follows:

- We propose **RESTful-Llama**, a framework that seamlessly integrates open-source language models with and adapts to any existing REST software system. This framework eliminates the long-term dependence on proprietary LLMs like GPT-4 (Achiam et al., 2023) and Claude (Anthropic, 2023), mitigating data privacy concerns and unlocking broader applications across various industries.
- We introduce **DOC_Mine**, a methodology

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 1433–1443

that instructs LLMs to generate a more diverse fine-tuning dataset from public REST API documentation. This method significantly enhances model fine-tuning across different contexts and improves success rates within the framework. Additionally, we release a new dataset containing 29,968 samples generated through DOC_Mine, empowering academia and industry to fine-tune models that follow the RESTful-Llama workflow[1] without relying on proprietary models.

- Experiments of RESTful-Llama on a dataset of 400 real-world REST API queries show a 31.9% improvement in robustness, and a 2.33x increase in efficiency, compared to the ReAct method.

2 Preliminaries

2.1 Related Works

Transformers and LLMs Transformers (Vaswani, 2017) have transformed numerous NLP subfields, such as text summarization (Ji et al., 2024), few-shot learning (Li et al., 2023b), adversarial robustness (Chen et al., 2024), information extraction (Li et al., 2023a), social computing (Liu et al., 2023), and question-answering (Xu et al., 2024). Their ability to capture long-range dependencies through self-attention mechanisms has significantly enhanced context comprehension, leading to higher accuracy across a variety of use cases. Recently, LLMs have further extended these capabilities and moved beyond traditional NLP applications. A key development is the ability of LLMs to interact with external systems, enabling them to tackle complex problems and integrate with real-world applications.

LLMs with external models: Recent research has explored connecting LLMs to various external models to address complex tasks. HuggingGPT (Shen et al., 2023) uses ChatGPT as a controller to perform task planning and select available Hugging Face models based on function descriptions. Visual ChatGPT (Wu et al., 2023) enables interaction between ChatGPT and multiple Visual Foundation Models (VFMs), allowing the exchange of images during conversations. GPT4Tools (Yang et al., 2023) adopts self-tuning to train open-source models to use tools to solve visual problems.

LLMs with tools and APIs: Another line of research has aimed at enhancing LLMs' proficiency in utilizing tools, allowing them to retrieve up-to-date information and perform operations by interacting with external tools. Chameleon (Lu et al., 2023) uses GPT-4 (Achiam et al., 2023) as a planner to coordinate a broad set of tools, such as web search engines and Python functions. ReAct (Yao et al., 2023) allows LLMs to interact with external environments like Wikipedia or ALFWorld (Shridhar et al., 2021) to solve general tasks. Gorilla (Patil et al., 2023) utilizes Llama 2 (Touvron et al., 2023) to solve machine learning tasks, but is limited to a set of 1,645 APIs selected from HuggingFace, Torch Hub, and TensorFlow Hub.

Table 1: Works that Connect LLMs with APIs/Tools/-Models.

Model	Number
Chameleon	13
Gorilla	1645
GPT4Tools	31
Visual ChatGPT	22
ReAct	4[2]
RESTful-Llama (ours)	**25,000+**

However, as shown in Table 1, current methods are constrained to a limited set of APIs, tools, or language models. Integrating arbitrary open-source LLMs with a widely used protocol like REST APIs offers a promising solution for expanding the scope of real-world applications.

2.2 RESTful APIs

Representational State Transfer (REST) APIs are a standard in web service development and are widely adopted across industries. According to the 2023 Postman API report (Postman, 2023), 86% of more than 40,000 developers and API professionals report using REST APIs. These APIs are based on the REST architecture, which uses standard HTTP methods (GET, POST, PUT, DELETE) to facilitate communication between clients and servers, while maintaining stateless interactions. When a client requests a resource through an endpoint, the server typically responds with data in JSON format, along with HTTP status codes such as 200 (success), 400 (client error), or 500 (server error) to indicate the outcome of the request (Masse, 2011).

Integrating LLMs with RESTful APIs opens ac-

[1]The fine-tuning dataset and the fine-tuned Llama 3.1-8B model are available at https://github.com/wmd3i/RESTful-Llama.

[2]The ReAct implementation supports 4 specific tasks, but the approach can be generalized to other tasks.

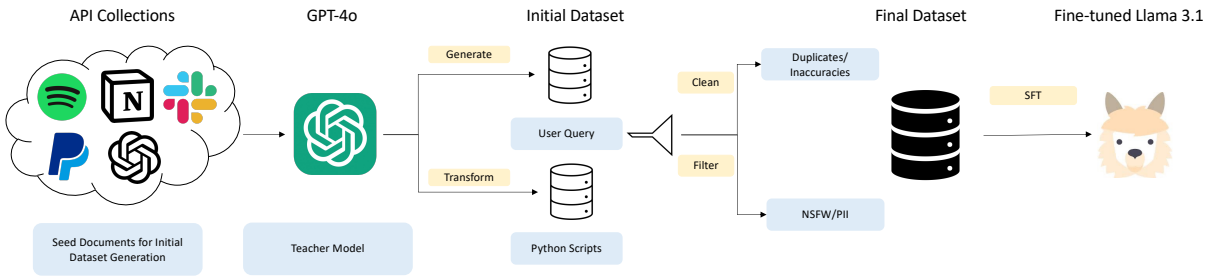

Figure 1: Data creation and fine-tuning steps

cess to a vast ecosystem of over 25,000 APIs[3], This aligns with industry practices, where the majority of developers and API professionals rely on REST architecture for their services (Postman, 2023). Moreover, using RESTful APIs provides enhanced control over potential risks, as the API endpoints are fully controllable. Therefore, it is essential for LLMs to not only adapt to a wide range of API systems, but also ensure robust and seamless interactions with them.

3 Dataset Creation and Fine-tuning

Given the widespread use and availability of REST API documentation, creating a diverse and high-quality dataset for fine-tuning open-source LLMs is essential. In this section, we outline the steps involved in dataset creation and fine-tuning. As illustrated in Figure 1, the procedure includes: (i) using the DOC_Mine approach to generate the dataset from public API documentation, (ii) cleaning and filtering the generated data, and (iii) the supervised fine-tuning (SFT) process.

3.1 DOC_Mine Approach

The DOC_Mine approach follows a backward generation strategy. Initially, an advanced teacher model[4] is prompted to generate Python REST API scripts from public API documentation. This process aligns with the LLM output in Section 4.4. Subsequently, the teacher model is prompted to reverse-translate these scripts into the natural language user queries, mirroring the inputs discussed in Section 4.2. A trace of the DOC_Mine workflow, including the prompting template, is outlined in Appendix D.

3.1.1 From API Documentation to Python Script

DOC_Mine leverages publicly available REST API documentation as seed data to generate Python scripts. We utilize API documentation from various industries, including Spotify (music), Notion (note-taking), Slack (communication), Paypal (payments) and OpenAI (AI). By drawing from these diverse sources, we generate a wide variety of Python REST API scripts and corresponding natural language user queries.

For each API, we identify and collect seed documents from Postman collections[5], which are primarily based on the OpenAPI Specification (OAS) JSON schema. This approach eliminates the need for explicit web scraping and conserves tokens when querying the teacher model. Using the teacher model, we generate Python scripts that incorporate user authorization tokens. However, these scripts may contain noise, such as inaccuracies and redundancies. To address this, we perform an additional filtering step to remove inaccurate or redundant data, ensuring that the fine-tuning process is not impacted by misleading information.

3.1.2 From Python Script to User Query

Next, we prompt the teacher model to translate the generated Python scripts into potential human natural language commands for API usage. In addition to common categories like gender, user mood, and age group, we specifically target an international audience by modeling major English dialects (Trudgill and Hannah, 2013). This allows us to capture user queries and behaviors across diverse situations. The template in Appendix D is populated with a variety of user groups and contexts, as detailed in Appendix A. By modeling different user groups and contexts, we ensure that DOC_Mine is aligned with varied user needs and scenarios.

[3]The number is reported by RapidAPI (RapidAPI).

[4]We use GPT-4o (OpenAI, 2024b) as the teacher model in this work.

[5]https://www.postman.com/collection/

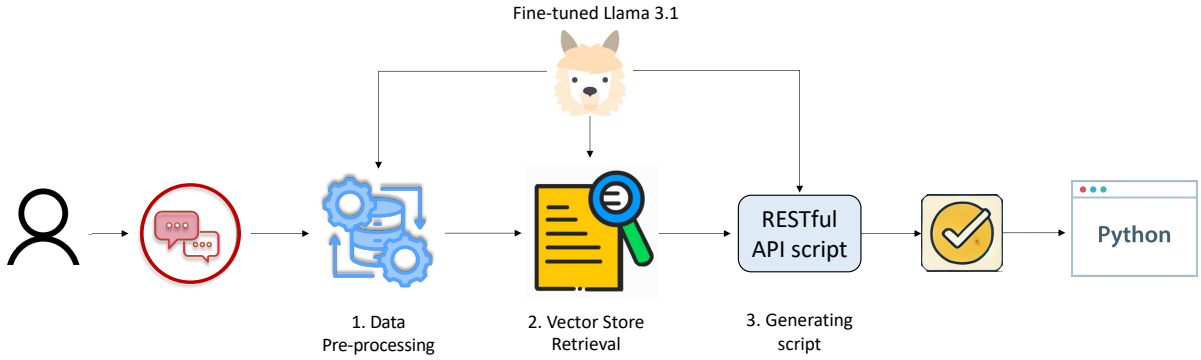

Figure 2: RESTful-Llama workflow

3.2 Data Cleaning and Filtering

To ensure the quality of the dataset, we first execute the generated python script and discard those that lead to errors or exceptions. Next, we perform data cleaning to remove duplicate API scripts and user queries from the generation. This step eliminates samples that are either identical or contain the same code snippets. Afterward, we apply an NSFW text classifier [6] to filter out any data that are not safe for work (NSFW), and use regular expressions (regex) to exclude data containing personally identifiable information (PII). Following the de-duplication and filtering steps, we finalized a dataset of 29,968 samples for fine-tuning. Detailed statistics on the cleaning and filtering process are provided in Appendix B.

3.3 Supervised Fine-Tuning (SFT)

We employ an SFT approach using the Llama 3.1-8B Instruct model (Dubey et al., 2024). This process takes a sequence $(t_1, t_2, \ldots, t_T)$, consisting of model inputs and outputs. The training objective is to minimize the standard cross-entropy loss $\mathcal{L}$, defined as:

$$\mathcal{L} = -\frac{1}{T}\sum_{i=1}^{T} \log P(t_i \mid t_1, t_2, \ldots, t_{i-1}),$$

where $P(t_i \mid t_1, t_2, \ldots, t_{i-1})$ represents the probability of the i-th token t_i, given the preceding tokens $t_1, t_2, \ldots, t_{i-1}$.

We utilize a single Nvidia A100 40G GPU and employ the LoRA technique (Hu et al., 2022) to perform SFT on the base Llama 3.1-8B Instruct model. LoRA adapters are applied to all linear layers of the model. Details of the training hyperparameters are provided in Appendix C.

[6] https://huggingface.co/michellejieli/NSFW_text_classifier

4 The RESTful-Llama Framework

As illustrated in Figure 2, RESTful-Llama consists of four main steps: (1) initialization with API documentation ingestion, (2) extracting essential information from the user query, (3) identifying relevant API documentation stored in the vector store, and (4) translating the user query into a Python script using the fine-tuned Llama 3.1-8B model, as described in Section 3.3. The complete workflow is summarized in Algorithm 1.

4.1 Initialization

During initialization, the workflow ingests REST API documentation for specific API systems into the vector store. We use the bge-small-en embedding model (Xiao et al., 2023) to vectorize the documentation. Each entry is treated as a vector store documentation node d_i, with an accompanying succinct description stored as its metadata.

4.2 Data Pre-Processing

In this step, the Llama 3.1 model extracts essential parameters from the user query, such as the address to be queried. These parameters are critical for the correct execution of REST APIs. Once extracted, they are forwarded to the script generation phase (Section 4.4), where they are reiterated to improve accuracy (Li et al., 2024).

4.3 Retrieval from Vector Store

We compute the cosine similarity between the user query and the API documentation nodes in the vector store. To ensure consistency, the user query is vectorized using the same bge embedding model as the stored nodes. The cosine similarity is calculated using the following formula:

$$cos\_sim(q, d_i) = \frac{q^T d_i}{||q|| \cdot ||d_i||}, \quad (1)$$

Algorithm 1: RESTful-Llama Workflow

Input : vector store *VStore*, embedding model *EmbM*, k, user query *query*
Output : Python script *script*

```
// Extract essential params
params ← pre_process(query);
// Query the vector store
query_embedding ← EmbM.get_embedding(query);
top_k_nodes ← VStore.query(query_embedding, k);
identified_node ← select_best_node(top_k_nodes, query);
llm_response ← generate_rest_api_script(identified_node, query, params);
script ← extract(llm_response);
script ← correct_params(script, params);
if validate(script) then
    return script;
else
    raise Error("Validation failed");
end
```

where q and d_i are the vector representations of the user query and API nodes. Based on these cosine similarities, the system retrieves the top-k[7] nodes. Given that a single REST service often includes numerous API endpoints, the vector store enhances efficiency by narrowing the candidate selection to the top-k most relevant APIs. The Llama 3.1 model is then employed to refine the selection process and identify the most relevant node.

As outlined in Algorithm 2, our approach concatenates the descriptions of the top-k retrieved nodes to form a comprehensive query. This query, along with a template prompt, is passed to the Llama 3.1 model. The model analyzes the input and selects the node most relevant to the user query using a predefined answering template. A helper function, get_best_node_idx(), is used to extract the index of the selected node. If the extraction fails, it defaults to the highest-ranked node (the node with the highest cosine similarity according to the retrieval). Finally, the user query and the selected documentation are forwarded to the next phase.

[7] k is a configurable parameter that can be tuned to the user's needs.

Algorithm 2: Node Selection Procedure (select_best_node)

Input : top-k nodes *top_k_nodes*, workflow template *template*, user query *query*
Output : Identified node *identified_node*

```
query ← template + query
foreach node in top_k_nodes do
    Append node's description to query
end
resp ← generate_best_node_resp(query)
idx ← get_best_node_idx(resp, k_threshold)
if idx is valid then
    identified_node ← top_k_nodes[idx]
else
    identified_node ← top_k_nodes[0]
end
return identified_node
```

4.4 Generating Python Script

Once retrieval is complete, the user query, parameters, and selected API documentation are organized and passed to the Llama 3.1 model. Leveraging retrieval-augmented generation (RAG) (Lewis et al., 2020), the model generates an executable Python script that interfaces with the desirable REST API. Python is chosen over other programming languages, such as Shell script, due to its greater flexibility in integrating with various libraries and systems, as well as its robust error-handling capabilities through try-except blocks.

Additionally, we have observed that the model occasionally misspells credential strings, especially when they are long. To address this, an interpolation step is performed to validate and correct any misspelled parameters by referencing the extracted parameters from Section 4.2. Finally, a syntax check is conducted to ensure the script compiles correctly. If the script fails this check, an error is raised.

5 Experiment

In this section, we evaluate the effectiveness of our approach, RESTful-Llama, compared to the baseline ReAct method. We also conduct case studies to investigate the impact of varying the k value in vector store retrieval.

5.1 Experimental Setup

Hardware To ensure smooth execution with sufficient GPU memory, we utilize a single Nvidia A100 40GB GPU. Both the Llama and Mistral (Jiang et al., 2023) models are run in bfloat16 (BF16) mode, offering a balance between accuracy and memory efficiency.

Datasets and Task To evaluate the performance of RESTful-Llama, we source eight *out-of-training-distribution* API categories from RapidAPI, including common real-world APIs such as Zillow, Urban Dictionary, Yahoo Finance, Booking.com, Twitter, NBA API, Google News, and Steam. We conduct a survey with two REST API users to gather 400 user queries related to these APIs. The task is defined as translating each user query into an executable Python script for the corresponding REST API call.

Evaluation Metrics For each task, we prepare a solution Python script that performs the intended action of the user query. Both the actual and expected Python scripts are executed. We then compare the REST API status codes and the field values in the response payload against the expected outcomes. A successful translation is defined as one where both the status codes and payload values from the actual script match those from the expected script.

We benchmark RESTful-Llama performance against ReAct with Llama 3.1-8B instruct model (baseline), which is adapted to the same REST API query task. For a fair comparison, we use the same vector store for both our approach and the baseline. To process the collected user queries, we configure the vector store retrieval with the top-5 results (i.e., $k = 5$) and import the corresponding RapidAPI documentation.

5.2 Results

Table 2: Success Rate (SR) and Average Time Comparison of Different Methods (Llama 3.1-8B)

Method	Vector Store Retrieval SR	Final SR	Avg Time ± Std (s)
RESTful-Llama (w/ fine-tuning)	**0.98**	**0.95**	**9.01** ± 1.13
RESTful-Llama (no fine-tuning)	0.91	0.85	11.48 ± 1.68
ReAct	0.86	0.72	21.03 ± 21.88

Table 2 provides a comparison of success rates (SR) and average task time across different methods using Llama 3.1-8B. The results highlight the performance of RESTful-Llama with Llama 3.1-8B fine-tuned on the DOC_Mine-generated dataset, RESTful-Llama using the vanilla Llama 3.1-8B, and ReAct paired with the vanilla Llama 3.1-8B. The table also reports vector store retrieval SR, as errors in retrieval often result in incorrect REST API endpoints and unexpected outcomes.

With fine-tuning on the DOC_Mine-generated dataset, RESTful-Llama's vector store retrieval SR improves by 7.7%, and the final SR increases by 11.7%. Although the APIs used in testing are out-of-training distribution and unrelated to the fine-tuning dataset, we hypothesize that fine-tuning enhances the model's understanding of REST API documentation, thereby improving its vector store retrieval SR. Beyond these gains, a deeper analysis shows that the fine-tuned model adheres more closely to RESTful-Llama's template and workflow, which further enhances its robustness.

Furthermore, RESTful-Llama achieves a 31.9% higher final SR compared to the ReAct method with its final SR of 95%, demonstrating its reliability in converting user queries into REST API calls. In contrast, an analysis of ReAct's errors highlights its struggles with extracting critical information and issues related to hallucinations.

Regarding the runtime, RESTful-Llama takes an average of 9.01 seconds to convert a user query into an executable Python script, compared to 21.03 seconds for ReAct, demonstrating a 2.33x speed improvement. Additionally, RESTful-Llama reduces the standard deviation by 19.36x, indicating much more consistent query response times. This efficiency is largely due to RESTful-Llama's reduced reliance on a prolonged prompt prefix, which contributes to more stable and predictable user wait times.

We also measured and compared the throughput, initialization (init) latency, and maximum GPU memory consumption for each method. Throughput is defined as the number of tasks completed per second, while init latency refers to the time required to complete the first task.

Table 3: Comparison of Other Metrics between RESTful-Llama and ReAct (Llama 3.1-8B)

Method	Throughput (task/sec)	Init Latency (sec)	Max GPU Memory (GB)
RESTful-Llama	**0.11**	**16.49**	**15.8**
ReAct	0.048	23.17	33.6

Table 3 demonstrates that RESTful-Llama delivers a 2.29 improvement in throughput and a

1.41x reduction in latency compared to ReAct. Additionally, RESTful-Llama uses just 15.8 GB of GPU memory, which is 53% less than ReAct. This increased efficiency, along with significantly lower memory consumption, makes RESTful-Llama more cost-effective and practical for real-world deployment.

Table 4: Success Rate (SR) and Time Comparison of Different Methods (Mistral-7B-v0.3)

Method	Vector Store Retrieval SR	Final SR	Avg Time ± Std (s)
RESTful-Llama (w/ fine-tuning)	0.87	0.77	**14.47** ± 2.40
RESTful-Llama (no fine-tuning)	**0.93**	**0.88**	15.40 ± 2.24
ReAct	0.91	0.85	65.59 ± 66.04

Lastly, we tested RESTful-Llama with the Mistral-7B v0.3 model (Table 4). Interestingly, the fine-tuned version of RESTful-Llama did not outperform the non-fine-tuned or ReAct methods in this case. We hypothesize that this is due to the DOC_Mine fine-tuning dataset not aligning well with the Mistral template. However, RESTful-Llama with no fine-tuning still demonstrates some improvement in accuracy over ReAct, and its processing time is significantly faster.

5.3 Ablation Study

Table 5: Success Rates and Time Comparison for Different k Values (Llama 3.1-8B)

Name	Vector Store Retrieval SR	Final SR	Avg Time ± Std Time (s)
k=1	0.90	0.88	**9.00** ± 1.42
k=5	**0.98**	**0.95**	9.01 ± 1.13

As shown in Table 5, we assess the benefit of using the model to select the best node from the top-k nodes, compared to directly choosing the node with the highest cosine similarity to the user query (i.e., $k = 1$).

Directly using the node with the highest cosine similarity significantly degrades RESTful-Llama's performance due to the high vector store retrieval error rate. Instead, $k = 5$ offers a much higher SR with only a minimal increase in average task execution time.

6 Industrial Application

In this section, we briefly describe how RESTful-Llama is adapted and deployed in the real world.

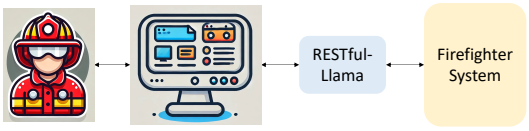

Figure 3: RESTful-Llama with fire department system

As shown in Figure 3, RESTful-Llama is integrated into a fire department system to assist firefighters in their operations. This integration enables individuals with no technical background and minimal training to effectively use the system. Users can submit queries, which RESTful-Llama translates into system commands, displaying the results back to the users.

7 Conclusion

In this paper, we introduce RESTful-Llama, a novel framework that bridges the gap between natural language processing and RESTful API operations for real-world applications. Specifically, we fine-tune the Llama 3.1 model to better align with the RESTful-Llama framework, which subsequently generates Python scripts that invoke the desired REST APIs according to user queries. Our experimental results demonstrate that RESTful-Llama outperforms existing methods, improving both robustness and efficiency in API interactions.

Limitations

While RESTful-Llama shows significant performance improvements over previous methods, its scalability in managing concurrent API calls on a large scale remains untested. Future work should explore optimizations in the underlying serving infrastructure or configurations to enhance throughput and handle larger volumes of requests more effectively.

Additionally, the current implementation's ability to robustly handle errors or unexpected inputs is still underdeveloped, and this remains an area for future improvement. Another potential enhancement involves allowing the framework to memorize historical user queries, which could enable two strategies: retrieving cached responses to speed up processing or learning from past correct and incorrect responses using techniques like reinforcement learning. Addressing these limitations will be the focus of our future work.

Ethical Considerations

This section outlines the ethical considerations involved in the research presented in this paper, with a particular focus on data privacy, security, and fairness.

- **Data Privacy and Security**: RESTful-Llama and DOC_Mine only use publicly available REST API documentation and avoid proprietary or sensitive data. Furthermore, any generated dataset with PII or NSFW content is filtered out. Additionally, both GPT-4o for dataset generation and Llama 3.1 for REST API script generation comply with data privacy standards.
- **Bias Mitigation**: To ensure fairness, we employ strategies to minimize bias in the dataset. DOC_Mine sources API documentation from diverse industries to reduce bias toward any specific domain. Additionally, we include various international user contexts, such as different English dialects, to ensure that generated queries are inclusive and applicable to a wide range of user groups.
- **Risk Control and Accountability**: RESTful-Llama operates in a controlled environment by generating only REST API scripts, which provide greater control over potential risks since API endpoints are fully manageable. The system relies on well-documented, publicly available APIs, enabling users to easily understand and monitor the process. However, as the generated results may be unpredictable, users should carefully evaluate potential risks before deployment.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2023. Introducing claude.

Zhuotong Chen, Zihu Wang, Yifan Yang, Qianxiao Li, and Zheng Zhang. 2024. Pid control-based self-healing to improve the robustness of large language models. *arXiv preprint arXiv:2404.00828*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yuelyu Ji, Zhuochun Li, Rui Meng, Sonish Sivarajkumar, Yanshan Wang, Zeshui Yu, Hui Ji, Yushui Han, Hanyu Zeng, and Daqing He. 2024. Rag-rlrc-laysum at biolaysumm: Integrating retrieval-augmented generation and readability control for layman summarization of biomedical texts. *arXiv preprint arXiv:2405.13179*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Sha Li, Ruining Zhao, Manling Li, Heng Ji, Chris Callison-Burch, and Jiawei Han. 2023a. Open-domain hierarchical event schema induction by incremental prompting and verification. *arXiv preprint arXiv:2307.01972*.

Shuoqiu Li, Han Xu, and Haipeng Chen. 2024. Focused react: Improving react through reiterate and early stop. In *Eighth Widening NLP Workshop (WiNLP 2024) Phase II*.

Zhuochun Li, Khushboo Thaker, and Daqing He. 2023b. Siakey: A method for improving few-shot learning with clinical domain information. In *2023 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 1–4. IEEE.

X. Liu, R. Wang, D. Sun, J. Li, C. Youn, Y. Lyu, J. Zhan, D. Wu, X. Xu, M. Liu, X. Lei, Z. Xu, Y. Zhang, Z. Li, Q. Yang, and T. Abdelzaher. 2023. Influence pathway discovery on social media. In *2023 IEEE 9th International Conference on Collaboration and Internet Computing (CIC)*, pages 105–109, Los Alamitos, CA, USA. IEEE Computer Society.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *The 37th Conference on Neural Information Processing Systems (NeurIPS)*.

Mark Masse. 2011. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.".

OpenAI. 2024a. Chatgpt.

OpenAI. 2024b. Introducing gpt-4o and more tools to chatgpt free users.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *Preprint*, arXiv:2305.15334.

Postman. 2023. 2023 state of the api report.

PyTorch. 2023. Pytorch hub.

RapidAPI. Rapidapi hub. `https://rapidapi.com/hub`. Accessed: 2024-06-25.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Peter Trudgill and Jean Hannah. 2013. *International English: A guide to the varieties of standard English*. Routledge.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2023. C-pack: Packaged resources to advance general chinese embedding. *arXiv preprint arXiv:2309.07597*.

Han Xu, Jingyang Ye, Yutong Li, and Haipeng Chen. 2024. Can speculative sampling accelerate react without compromising reasoning quality? In *The Second Tiny Papers Track at ICLR 2024*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. GPT4tools: Teaching large language model to use tools via self-instruction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

A Contextual Parameters

Table 6: Contextual Parameters for Tailoring Prompts

Context Category	Groups
Gender Group	Male, Female, Non-binary
User Mood	Happy, Sad, Angry, Relaxed
English Dialect	American, British, Australian, New Zealand, South African, Indian
Age Group	Teen, Adult, Senior

B Data Cleaning and Filtering Statistics

Table 7: Data Cleaning and Filtering Overview

Name	Initial Count	Filter Count	Final Count
Spotify	38,016	28,412	9,604
Notion	9,936	7,764	2,172
Slack	50,976	39,194	11,782
PayPal	45,360	41,129	4,231
OpenAI	13,392	11,213	2,179
Total	157,680	127,712	**29,968**

C Training hyper-parameters

Table 8: Fine-tuning Hyper-parameters

Parameter	Value
Learning Rate	1.0e-4
Training Epochs	3.0
LR Scheduler Type	cosine
Warmup Ratio	0.1
Precision	BF16
LoRA Rank	8
LoRA Alpha	32
Dropout Rate	0.1

D DOC_Mine Trace

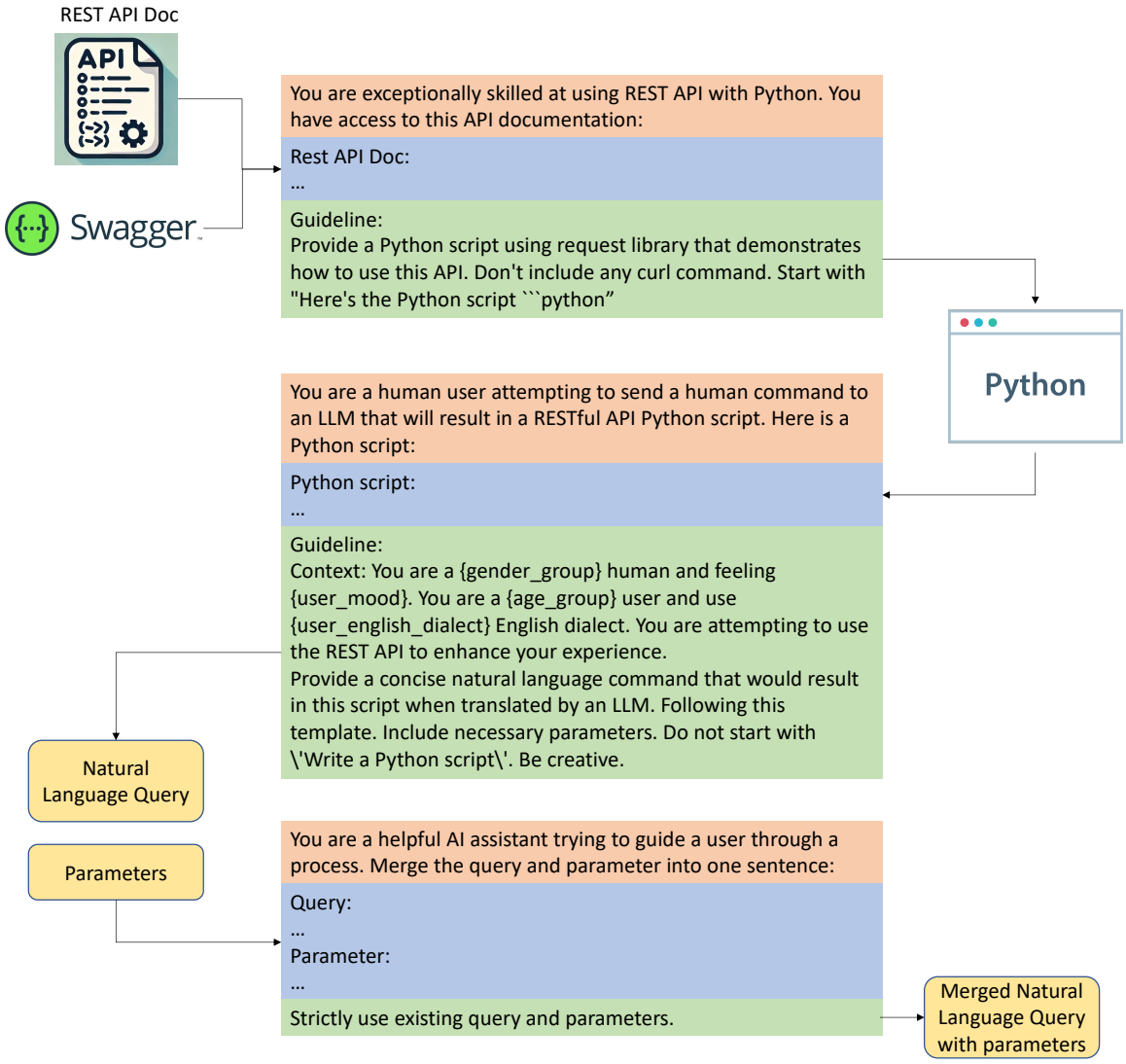

Figure 4: DOC_Mine Trace - Blue areas are placeholders to be filled in within the template. Orange boxes provide context, while green boxes contain instructions.