

Hard-Routed Adapters: Enhancing API Calls in Multi-Agent LLM Pipelines

Patryk Kuchta



Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2024

Abstract

The market proliferation of voice assistants (Canalys, 2021), has encouraged the investigation of Tool-Use capabilities powered by Large Language Models (LLMs). Qin et al., 2023 utilized a fine-tuned Llama-2 model (Touvron et al., 2023) in a ReACT (Yao et al., 2023) prompting context for Tool-Use, which has been shown to struggle with populating the API calls with arguments (Q. Xu et al., 2023). Shen et al., 2024 proposed improving the pipeline through the separation of the ReACT reasoning stages into three models: Planner, Caller and Summariser. This still pertains to the reliance on zero-shot comprehension of APIs, which is a weak assumption given the models used and research investigating zero-shot abilities of models of this size (Hsieh et al., 2023; Wei et al., 2022). This study addresses this weak assumption and uses a hard routing-based strategy for utilizing expert adapters to significantly boost the quality of the API Calls produced by the Caller model, by reducing the zero-shot capability reliance. Heuristics regarding the application of this idea were investigated to derive the optimal level of abstraction, number of tasks and samples. Furthermore, this study shows that adapter aggregation is a viable strategy for lowering the computational cost of adapting to changes in the APIs. The experimentation has shown that this pipeline can allow the 7 Billion model, to beat much larger, modern and closed-source models used in a zero-shot scenario on this task.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Patryk Kuchta)

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Pasquale Minervini, whose invaluable input, unwavering support, and dedication have been instrumental throughout this journey. His guidance and feedback have greatly enhanced the process of writing this dissertation and made it a more educationally enriching experience. For his patience, encouragement, and steadfast belief in my abilities, I am truly thankful.

I would also like to take this moment to thank all the friends who have provided their feedback on various parts of my dissertation. Your thoughtful insights and constructive critiques have been instrumental in shaping this work. I am particularly grateful to Simone Bartoli, Umut Halil, Beāte Desmitniece, Elias Forstrøm Fyksen and Alexander Angelov for their specific contributions and encouragement. Your support has been truly appreciated.

This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF) (<http://www.ecdf.ed.ac.uk/>).

Table of Contents

1	Introduction	1
1.1	Motivation for Tool-Use	1
1.2	Hypothesis and Approach	2
1.3	Aims	4
1.4	Report Structure	5
1.5	Objectives	6
2	Literature Review	7
2.1	Background	7
2.1.1	Large Language Models	7
2.1.2	Augmenting LLMs with Tool-Use	9
2.1.3	Parameter-Efficient Fine-Tuning	11
2.2	Related Work	12
2.2.1	Non-ReACT Tool-Use	12
2.2.2	Multi-Agent Tool-Use	13
2.2.3	Progressive Fine-Tuning	14
3	Methodology	16
3.1	Training Dataset	16
3.1.1	Source and Method of Generation	16
3.1.2	Expert-Wise Separation	17
3.1.3	Samples Distribution Distribution	18
3.1.4	Classifying the Tool Request	20
3.1.5	Certainty Separation	22
3.2	Test Sets	25
3.2.1	ToolBench Test Set	25
3.2.2	ToolAlpaca Test Set and Additional Annotation	26

3.2.3	Specific Developmental Test Set	26
3.3	Training Scheme Experiments	27
3.3.1	Selection of the Base Model	27
3.3.2	Aggregating Experts into Higher-Order Experts	27
3.3.3	Hierarchical Progressive Adapter Stacking	28
4	Results	30
4.1	Optimal Hierarchical Level of Abstraction	30
4.2	Effect of Sample Number and the Tasks Considered	31
4.3	Effect of Limiting the Dataset to Certain Plans	32
4.4	Preferred Base Model Selection	33
4.5	Viability of Expert Aggregation for API Experts	34
4.6	Progressive Adapter Stacking for Boosting Performance	35
4.7	In-Domain and Out-Of-Domain Test and Comparison	36
5	Conclusions	39
5.1	Limitations and Future Work	39
5.2	Summary	40
A	Model Glossary	47
B	Chart Data	49
B.1	Methodology chapter	49
B.1.1	Training Dataset	49
B.2	Results chapter	51
B.2.1	Optimal Hierarchical Level of Abstraction	51
B.2.2	Effect of Sample Number and the Tasks Considered	53
B.2.3	Effect of Limiting the Dataset to Certain Plans	54
B.2.4	Preferred Base Model Selection	55
B.2.5	Viability of Expert Aggregation for API Experts	56
B.2.6	Progressive Adapter Stacking for Boosting Performance	57
B.2.7	In-Domain and Out-Of-Domain Test and Comparison	58
C	Prompts Used	60
C.1	ToolAlpaca Test Set Categorization	60
C.2	ToolBench Multi-Agent prompt structure	62

Chapter 1

Introduction

1.1 Motivation for Tool-Use

The growth in computational and architectural capabilities has enabled the size expansion of Large Language Models (LLMs), allowing new capabilities to emerge (Brown et al., 2020; Wei et al., 2022). Amongst those previously infeasible tasks is Tool-Use (Hsieh et al., 2023). In their review, Z. Wang et al., 2024 describe it as providing the LLM with various utilities that are not part of the neural model, allowing the pipeline to perform previously impossible actions and improve the quality of the answer. Those actions can range from consulting a calculator to achieve better numerical answer precision (Schick et al., 2023), using a web browser to fetch current information that was not present during the training of the LLM (Zhuang et al., 2023), or even controlling an external robotic assistant through calls (Vemprala et al., 2023). Figure 1.1 includes a simulated Tool-Use conversation.

In principle, LLMs can easily leverage a large variety of tools through public Application Programming Interfaces (APIs). This ability is due to their programmatic rule-based nature, which strikes a large resemblance with code and natural language while generating text of this kind is within the realm of Natural Language Processing (NLP) solutions (Duque et al., 2023; Sarkar et al., 2022). The availability of various APIs is an essential aspect of the practical usefulness of the Tool-Use pipeline, as they can provide crucial help when dealing with a plethora of tasks. For instance, RapidAPI is an API hub used in some Tool-Use studies (Qin et al., 2023; Shen et al., 2024; Z. Shi et al., 2024), and it contains over 16000 endpoints, spanning over 49 categories. Hubs are preferable to using APIs directly as when used commercially they unify billing and documentation into one place, increasing their practical utility.

Some may argue that we have already seen the first wave of market adoption of Tool-Use through the proliferation of voice assistants such as Google Assistant or Amazon Alexa (Canalys, 2021). Question-answering (QA) systems can benefit from Tool-Use, as can be seen by the performance benefits of Retrieval Augmented Generation (RAG) (Asai et al., 2024; Guu et al., 2020; Izacard et al., 2022; P. Lewis et al., 2021), which can be considered a subset of Tool-Use. Those techniques are much more resilient in cases of information that was not present in the training data (Min et al., 2023) and in terms of data which values have changed since training (Izacard et al., 2022). The robustness provided by RAG is a crucial consideration in many systems, given the cost of retraining an LLM. Furthermore B  chard and Ayala, 2024; H. Ding et al., 2024 show that introducing information-retrieving tools into the pipeline reduces the frequency of hallucinations, further emphasising the benefits of making LLMs capable of using Tools.

1.2 Hypothesis and Approach

Current solutions based on small models, such as the Llama model, often fail due to incorrectly generated API calls. Q. Xu et al., 2023 highlight that API call malformation in the shape of incorrect Argument Population¹ is the most significant contributor to failures of the entire pipeline. The idea central to the improvements offered by the Multi-Agent pipeline (Shen et al., 2024) is to consider each of the reasoning steps of ReACT reasoning (Yao et al., 2023) used by Qin et al., 2023 as a separate Machine Learning problems, necessitating separate models for Planning, Calling and Summarising. This approach allows the Caller model to become more specialised in API calling. Figure 1.2 illustrates the Multi-Agent architectural shift compared to a Single-Agent approach and a non-Tool-Use conventional pipeline.

In this thesis, the hypothesis investigated is that each endpoint could be regarded as a distinct task. Similarly, entire API families² can be treated as a single unit, and categories³ comprising lists of API families can also be viewed as a unified task. This approach reduces the emphasis on the model’s ability to learn API usage solely from the documentation in the prompt, as it allows the model to gain more insight by

¹Argument Population is the task of selecting the parameters and values for those parameters in the API call.

²API families are sets of endpoints from the same route; for example, all endpoints accessible from <https://soundcloud4.p.rapidapi.com/> constitute the soundcloud API family.

³The creators of each API family annotate categories by selecting from a list of 49 categories defined by RapidAPI.

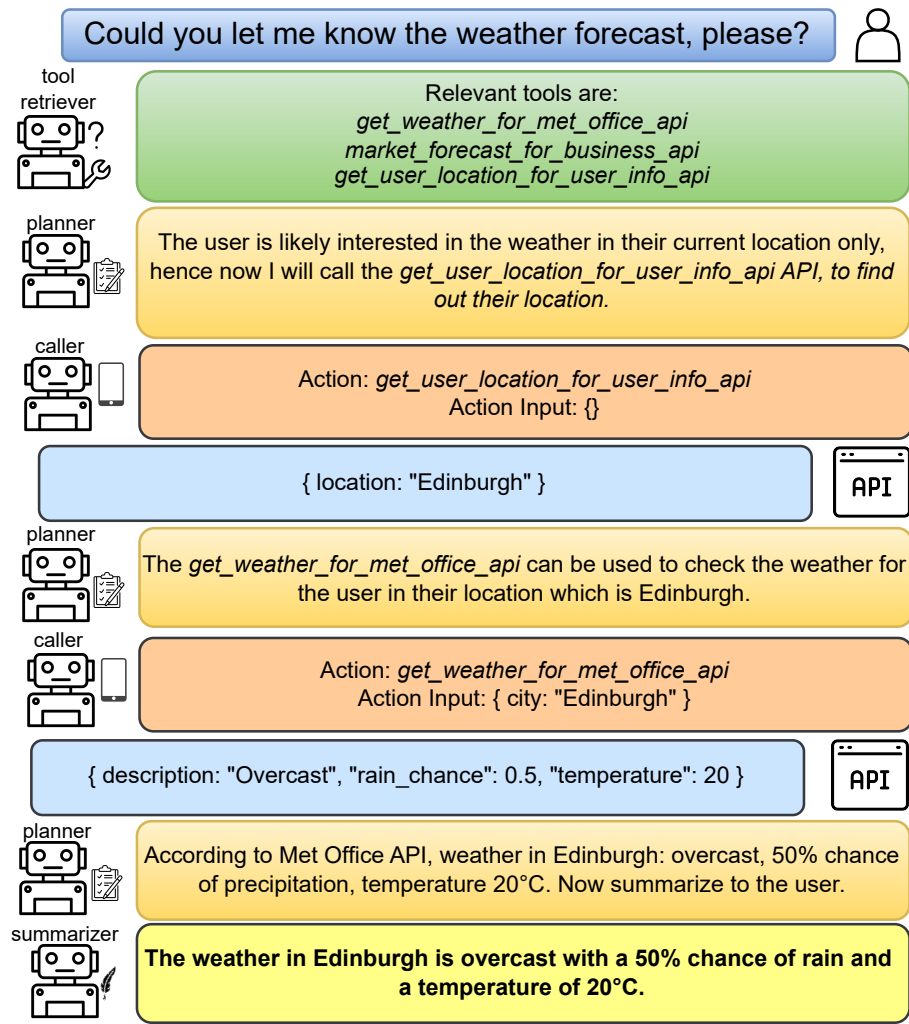


Figure 1.1: Simulated conversation following ReACT prompting principles (Yao et al., 2023), incorporating the task division proposed by Shen et al., 2024. Qin et al., 2023 employed the same model for the Planner, Caller, and Summarizer roles.

training on a dataset focused on a specific subset of the overall data. This hypothesis aligns with previous research suggesting that smaller models perform better when reliance on in-context learning is minimised through fine-tuning (Mosbach et al., 2023). This fragmentation of tasks allows for many possibilities in terms of implementation, generating several sub-hypotheses examined in this study.

In a practical scenario, procuring even a relatively small subset of Llama-2 7B (Touvron et al., 2023) experts through fine-tuning becomes infeasible, as storing an entire model for each expert would be highly inefficient in terms of storage. A Parameter Efficient Fine Tuning (PEFT) technique is used to maintain the computational and

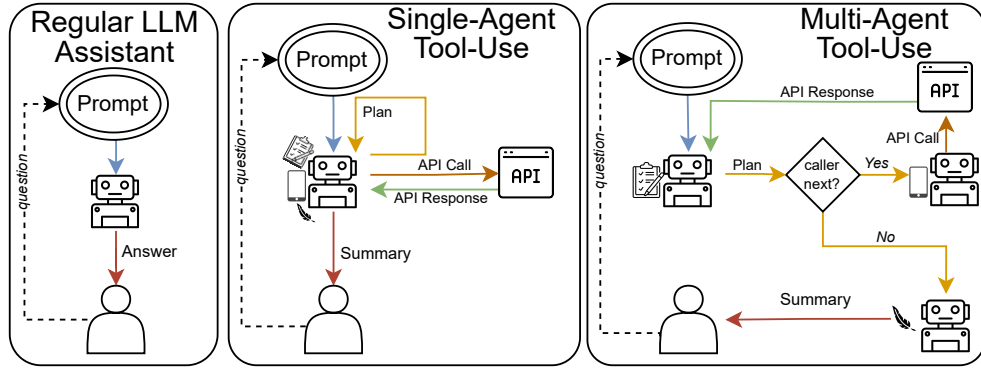


Figure 1.2: Illustration comparing the standard LLM assistant Question-Answering approach with the Tool-Use ReACT-based pipeline (Yao et al., 2023), in both Single-Agent (Qin et al., 2023) and Multi-Agent (Shen et al., 2024) configurations.

storage feasibility of the research undertaken. In many studies, including this one, the resulting weights from PEFT technique applications are called adapters⁴ which are applied onto a base model. In the context of the study, the PEFT strategy is Low-Rank Adapters (LoRA) (Hu et al., 2021) due to its wide adoption and the fact that it allows for the adapters to be dynamically swapped into the network efficiently. Hard routing with a library of LoRA adapters is used, based on the description by Ostapenko et al., 2024, where the pipeline dynamically swaps one adapter onto the base model based on the current requirements. In this scheme, backing off to the base version of the model is efficient, as it simply requires deactivating all of the adapters.

1.3 Aims

Crucially, the study aims to validate that this approach can improve performance beyond what was achieved by Shen et al., 2024. The proposed method has a high degree of customisation possible regarding base models, number of samples/tasks, adapter stacking and hierarchical selection. Hence, to evaluate how to make the most of this proposed pipeline, this research investigates the most appropriate hierarchical level of abstraction for training, given that for each tool, there are 3 task splits possible: endpoint-wise, API-family-wise, and category-wise. Furthermore, the experimentation approximates an optimal number of samples and tasks (endpoints considered) for training the experts.

⁴Adapters can be viewed as lightweight components that, once applied to a model, allow it to adapt to have new capabilities—more detail in Section 2.1.3

The hierarchical nature of the dataset division allows for the investigation of the effect of aggregating experts to produce a new expert from one level higher in the hierarchy, for instance, by using all of the endpoints experts from an API family to create an API family-wide expert, based on adapter aggregation that has already been shown to have a positive impact in Natural Language Generation (NLG) (Y. Wang, Agarwal, et al., 2022). This approach to building experts might be beneficial regarding computational cost as it simplifies expanding the number of endpoints supported in the future, as new experts can be aggregated with minimal expense.

The final approach relevant to this study is extending the idea of the Global-to-Local Progressive Fine-Tuning (GLPFT) presented by Shen et al., 2024 beyond two stages to test whether this training regime improves the final performance of the resulting API experts on the task. Prior research in other domains has shown that stacking PEFT adapters can positively impact performance (Gema et al., 2024) when the training regime successively narrows down the task.

1.4 Report Structure

This report is organized into five main chapters. The first chapter Introduction introduces the motivation, hypothesis, and research objectives of this study, upon which the next chapters are built. Chapter two is the Literature Review and it examines relevant background in the field of LLM-based Tool-Use and the research related to this study. This section also includes an introduction to many of the concepts used throughout this study. Methodology, which is the third chapter, outlines the experimental configuration and training schemes. Furthermore, it encompasses a discussion of the modifications and additional annotations that had to be made to the dataset, and it also touches on how the dataset was generated in the first place. Chapter four, called Results, discusses the findings, highlighting the impact of different factors on the performance of the pipeline. Finally, the Conclusions summarize the key insights, address limitations, and suggest future research directions. Additional tables and prompt templates are included in the Appendix. Figure A.1 in the Appendix also includes an illustration organizing all the models used in this study to provide clarity.

1.5 Objectives

- Validating the performance of the Multi-Agent (Shen et al., 2024) pipeline:
 - Replicating the results of Shen et al., 2024 to ensure the viability of the task-splitting hypothesis for the Tool-Use task.
 - Analysing the impact of specialised agent networks on the ability to produce valid API calls.
- Analysis of the ToolBench dataset (Qin et al., 2023)
 - Identifying potential issues and inconsistencies in the train and test dataset, especially in the relation between the Planner-Caller steps
 - Addressing inconsistencies in the dataset by automated removal
 - Analysing the effect of the training on the filtered dataset
- Implementation of the Modular-Expert-LLM API Caller pipeline:
 - Dividing the dataset into a finer set based on three hierarchical division principles: API-category-wise, API-family-wise, and API-endpoint-wise.
 - Implementing a pipeline for fine-tuning expert adapters for the fine tasks.
 - Implementing a modular inference pipeline that will use the experts to produce better API calls
- Finding the best principles and approaches for expanding the Modular-Expert library
 - Comparison of the hierarchical division principles
 - Combining finer level experts into coarser level experts using aggregation
 - Analysing the relation between the number of training samples available and tasks (endpoints) considered with performance
 - Experimentation with a stacked LoRA progressive fine-tuning approach
 - Comparison of the stacked progressive fine-tuning approach against a simple stacked adapter approach and direct fine-tuning
- Analysing the practical impact of the Modular-Expert LLM pipeline.
 - Validating performance gains of the inference pipeline on the original ToolBench (Qin et al., 2023) test set
 - Comparing the effectiveness of the proposed solution against closed source models GPT-3.5 Turbo and GPT-4o mini
 - Evaluating the use of categorical experts on out-of-domain tools⁵ from the ToolAlpaca (Tang et al., 2023), with automatic neural category annotation

⁵Out-of-domain tools are tools for which there are no training samples present in the training set.

Chapter 2

Literature Review

2.1 Background

2.1.1 Large Language Models

The invention of the transformer architecture, proposed by Vaswani et al., 2023, has allowed for an unprecedented growth of models that are modelling human language. Pre-trained Large Language Models such as BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019) ignited the first wave of adaptation of large language modelling technology by providing the reasoning capabilities required for complex classification tasks, such as sentiment analysis (Sun et al., 2019), named entity recognition (Y. Wang et al., 2020), and question answering (Gupta, 2020), with remarkable accuracy and efficiency.

Generating text can be treated as a repeated classification task, continuously choosing the next token until a special end-of-sequence symbol is produced. It is important to note that this scheme does not allow for corrections earlier in the sequence, which is a significant limitation (implications discussed in Subsection 3.1.4). This approach led to the creation of large Sequence-to-Sequence models such as the BERT-based BART (M. Lewis et al., 2019). Another family of models capturing this notion is the now-famous Generative Pre-trained Transformer (GPT) family of models. Brown et al., 2020 presented the third iteration from this collection, and with the increase in the number of parameters also came a massive growth in capabilities. One particularly impressive feat was that the models started to exhibit a considerable degree of zero-shot and few-shot capabilities (Wei et al., 2022). In NLP, zero-shot learning is the ability of the LLM to learn how to answer the task using instructions in the prompt alone

without seeing any question-answer pairs in the training. For example, LLM is trained to respond with sentiment scores after a comment is pasted into the conversation, but if the task is explained sufficiently before the comment it should have no issues consistently answering it. Appendix C.2 is a template that uses zero-shot learning for Tool-Use, and it is included along with an example. Few-shot learning extends this idea by providing a very limited number of samples and the task description in the prompt. Both approaches rely on the LLM adapting based on the information in the prompt, commonly known as in-context learning (Dong et al., 2024).

In modern Natural Language Processing, transfer learning (Torrey & Shavlik, 2009) has become a crucial concept. Transfer learning occurs when a fully trained model on a related task provides the initial weights for a new model for the target task. The two distinct stages of training in this approach are often referred to as pre-training, followed by fine-tuning. In most applications of this concept, it is hoped that the initialisation of the specialised models with this general model will aid the training beyond what is achievable by training from scratch or at least reduce the training time. The same idea is central to the current LLM research and applications since foundation models have reached the mainstream (Zhou et al., 2023).

Due to their computational complexity, the broader adoption of LLMs comes with a significant environmental and financial cost. The ecological burden is substantial in the current climate crisis, which appears to be at odds with furthering the development of LLM technology. The carbon and cost impact of the discussed technologies was the subject of several studies, and they offer rather pessimistic conclusions on the matter (Faiz et al., 2024; Patterson et al., 2021). In light of these challenges, improving the efficiency and effectiveness of smaller models becomes increasingly important.

Another large obstacle to practical applications of LLM is hallucinations, which are instances where language models generate plausible-sounding but baseless information in response to queries (Huang et al., 2023; Ji et al., 2023). This phenomenon is particularly prevalent in small models (Dahl et al., 2024) and has significant implications for practical applications, even garnering attention from non-computer science academics (Curran et al., 2023). There is a significant positive correlation between the extent of fine-tuning and hallucination rates (Gekhman et al., 2024), presenting additional challenges for small instruction fine-tuned models.

2.1.2 Augmenting LLMs with Tool-Use

In addition to enabling previously impossible operations and enriching answers, integrating external retrieval methods, such as Tool-Use, has shown promising results in reducing hallucinations (Bécharde & Ayala, 2024; H. Ding et al., 2024). The research conducted by Qin et al., 2023 was amongst the first to tackle the Tool-Use task in the context of smaller models. In their study, they experimented with multiple variants of models from the Llama-2 family (Touvron et al., 2023). Amongst them was the 7 billion parameter variant, which is relatively small.

Models of this size are lightweight enough to run smoothly on high-end consumer GPUs, offering instant responses in real-time applications. As mentioned before, models of this size possess limited abilities in zero-shot settings (Hsieh et al., 2023; Wei et al., 2022). Therefore, the model was instruction fine-tuned on a dataset of conversations with Tool-Use generated by GPT-3.5 Turbo (Brown et al., 2020) (more detail in Subsection 3.1.1). This process teaches smaller models to mimic the behaviour of larger ones and is often referred to as Distilling (Iliopoulos et al., 2022). The dataset features a diverse array of tools, many with a minimal number of samples (discussed in more detail in Subsection 3.1.3). This characteristic requires the models to demonstrate zero-shot learning capabilities for specific APIs.

This model retrieves a small subset of the endpoints in the RapidAPI hub, allowing for the inclusion of the most relevant API documentation within the limited context window. The success of the entire turn is highly dependent on whether the correct APIs get fetched, therefore the accuracy of that retriever is crucial. The role of tool retrieval in creating the prompt is demonstrated in Figure 2.1.

The maximum context length is a characteristic limitation of the transformer architecture (Vaswani et al., 2023) upon which modern LLMs are built (Brown et al., 2020; Jiang et al., 2023; OpenAI, 2024; Touvron et al., 2023). This consideration makes harnessing the benefit of few-shot approaches difficult and costly as Tool-Use conversations tend to be long, often surpassing 4096 tokens (which is the limit for Llama-2 models). In recent developments, context windows started rapidly growing (Y. Ding et al., 2024; G. Team et al., 2024; Xiong et al., 2023), which should allow the retrievers to be more liberal with the number of APIs they select and prevent conversation cropping when the context limit is reached. Some studies propose architectures that allow infinite tokens to fit into the context (Munkhdalai et al., 2024). This does not imply that the retrieval problem is wholly solved because, as F. Shi et al., 2023 shows, LLMs can get distracted

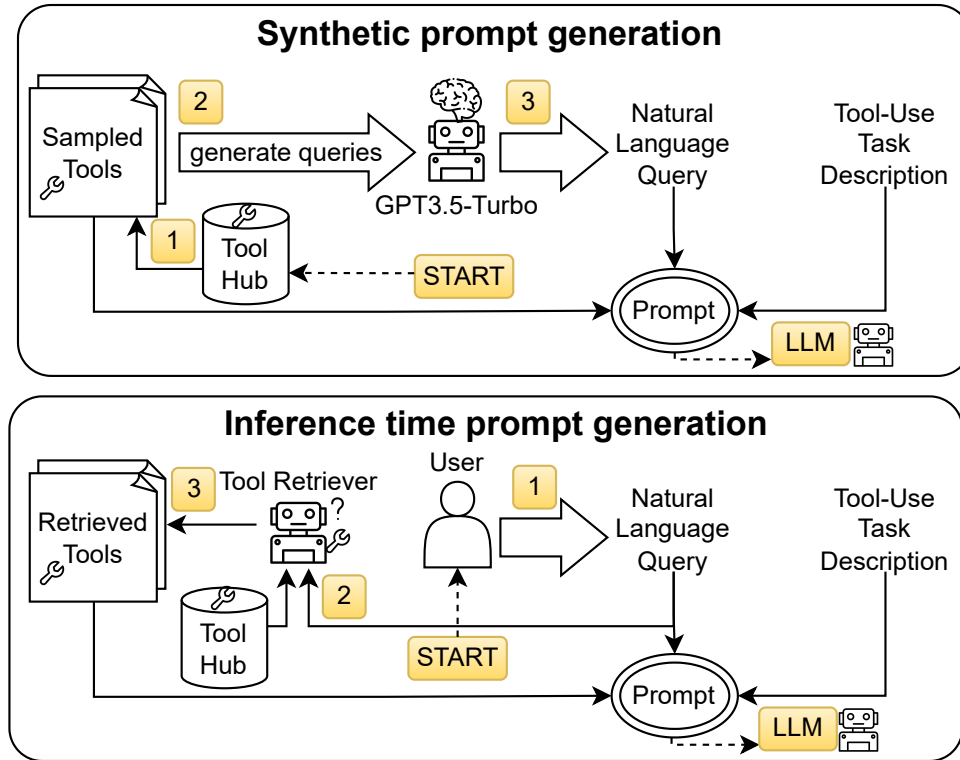


Figure 2.1: Comparison of synthetic and inference-time prompt generation. Synthetic prompt generation was utilized exclusively during dataset creation.

away from the task when irrelevant information is included in the prompt. Furthermore, previous studies have shown the position of information can have a massive impact on its utilisation (N. F. Liu et al., 2023). Qin et al., 2023 show that using LLMs with more oversized context windows without changing the behaviour of the retriever already has a substantial impact on the performance, simply by preventing context cropping.

Their study also considered the reasoning structure, where they proposed a Depth First Search-based Decision Tree (DFSDT) optimised to limit error propagation and encourage exploration during generation. Their evaluation has suggested the superiority of the DFSDT approach. This approach is an extension of the strategy commonly known as ReACT (Yao et al., 2023). In their study, they demonstrate that incorporating a reasoning step before each action and allowing the model to repeat this cycle multiple times can have a substantial benefit when addressing complex tasks and it is a significant improvement over previously proposed Chain-of-Thought (CoT) prompting (Wei et al., 2023).

The variety of APIs is part of what makes Tool-Use so practically useful. It also often requires the model to learn how to use a specific API based on a digest of the

documentation alone. This, in turn, means that there is a significant emphasis on the zero-shot abilities of the model, which in the case of the model considered is lacking (Hsieh et al., 2023; Wei et al., 2022). Q. Xu et al., 2023 show that in the case of Llama-based Tool-Use models, mistakes in the Argument Population stage are the most common factor contributing to failures. The focus of this study arises when taking both of these considerations into account, hence it investigates whether a shift towards a finer-task split can boost the performance, as it in principle reduces the reliance on zero-shot assumptions.

2.1.3 Parameter-Efficient Fine-Tuning

As large language models have grown in size and complexity, innovative techniques have become essential to manage their computational and storage demands effectively. This trend has led to the development of efficient adaptation techniques in the field of NLP (Houlsby et al., 2019; Semnani, 2019). Despite the large number of potential approaches (Han et al., 2024; L. Xu et al., 2023), LoRA (Hu et al., 2021) and its derivatives (Dettmers et al., 2023; Lialin et al., 2023; S.-Y. Liu et al., 2024) have started to be regarded as the state-of-the-art approach. The underlying scheme is simple and can be adapted to suit many scenarios, as it only involves training two small matrices, the dot product of which gets added to the original weights of any matrix in the model. The size of the matrices can be adjusted through the r parameter, which defines the hidden dimension of both matrices. In practical applications, applying this adaptation to only some of the weight matrices of the model works well enough (He et al., 2022), further increasing the appeal of this approach. The central equation of LoRA can be formulated in the following way:

$$f_{\text{adapted}}(x|W, A, B) = f(x|(W + AB)) = xW + xAB \quad (2.1)$$

$$\text{Adapter Parameters: } \mathbf{A} \in R^{d_1 \times r}, \quad \mathbf{B} \in R^{r \times d_2}, \quad r \ll d_1 \wedge r \ll d_2 \quad (2.2)$$

$$\text{Original Weights: } \mathbf{W} \in R^{d_1 \times d_2} \quad (2.3)$$

LoRA offers an extraordinary reduction in trainable parameters, reducing it to 1-3% of the original count while still maintaining good performance on various tasks (L. Xu et al., 2023), and only those trained parameters in the adapter matrices have to be saved, significantly reducing the storage requirements of the solutions involving multiple experts. Furthermore, as shown by Figure 2.2, LoRA can be used without

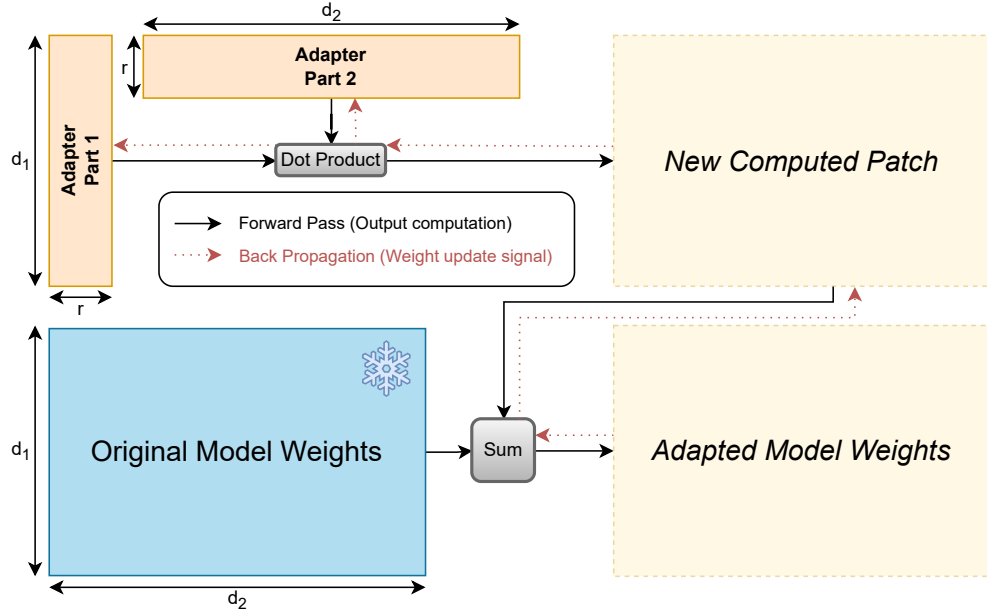


Figure 2.2: An interpretation of the effect of LoRA adapters on the final weights through a latent patch, while the original weights remain frozen (unchanged during training).

altering the original weights, allowing for dynamic swapping in the inference time and reducing the overhead of the solution. Furthermore, Biderman et al., 2024 shows that LoRA can prevent forgetting of the original abilities of the model when compared with traditional full fine-tuning, which is an advantageous characteristic when using a pre-trained model.

2.2 Related Work

2.2.1 Non-ReACT Tool-Use

Many approaches, including the pipeline considered in this paper, rely on ReACT prompting (Yao et al., 2023), but it is worth noting that there are other strategies as well. For instance, in Chain of Tools (Z. Shi et al., 2024), the reasoning is expressed as a program (for example, in Python) that includes a class of functions that list programmer-written programs that can call the tools. It offers a sizable performance improvement over other approaches. However, it inherently requires additional engineering as all of the APIs must be accessible through Python function calls. Further, in Chain-Of-Code (Li et al., 2023), reasoning is also expressed as Python code that then is interpreted jointly by the compiler and LLM, where the LLM serves as a backup for whenever the

compiler is unable to access a function that an LLM included in the code. An example of such a function would be `'is_fruit("lemon")'`, which is a question that the LLM can confidently answer with its natural language capabilities while answering this question programmatically or with API would be wasteful.

Schick et al., 2023 have proposed another approach to Tool-Use, where information that can be retrieved/computed by external tools is replaced by function calls that can return this information. This approach allows the model to simplify the generation process by using a single step to insert function calls for retrieving details needed in the sentence (refer to Figure 2.3 for an example). This scheme avoids the cost and complication of multi-step reasoning chains. It allows for the relatively easy creation of new datasets based on existing text, where the 'information gaps' are artificially introduced. However, the scheme has limited resilience against errors, as the one-step nature does not allow the model to correct its mistakes.

In all of the approaches discussed in this section, the writing of API calls is not done by an LLM but rather by a procedural function. Those approaches reduce the reliance on the LLM's ability to write API calls, which in the case of Llama-based solutions is especially poor (Q. Xu et al., 2023). However, they necessitate a much larger engineering effort to create a pipeline and integrate many tools. The implications of this difference become evident by comparing the number of tools Schick et al., 2023 and Li et al., 2023 used (5 and 107 respectively) with the portfolio of endpoints used in the Qin et al., 2023 study (over 16 thousand). Figure 2.3 is an illustration that compares the main characteristics of these schemes.

2.2.2 Multi-Agent Tool-Use

Shen et al., 2024 have considered that performing ReACT reasoning might be too complex for a single small model. Therefore, to reduce the reliance on this weak assumption, they fine-tune separate models on three distinct tasks: Reasoning (referred to as Planning in Shen et al., 2024), Acting (Calling), and Summarizing. This scheme allows the individual models to focus more on their respective tasks, which is helpful given their constraints.

While routing during training is a trivial problem, as the correct agent alignment is known apriori, during inference routing is a significant problem, as it is not apparent which agent should respond at each step. Therefore, Shen et al., 2024 trained the Planner to conclude their responses with `Next: caller` or `Next: summariser`,

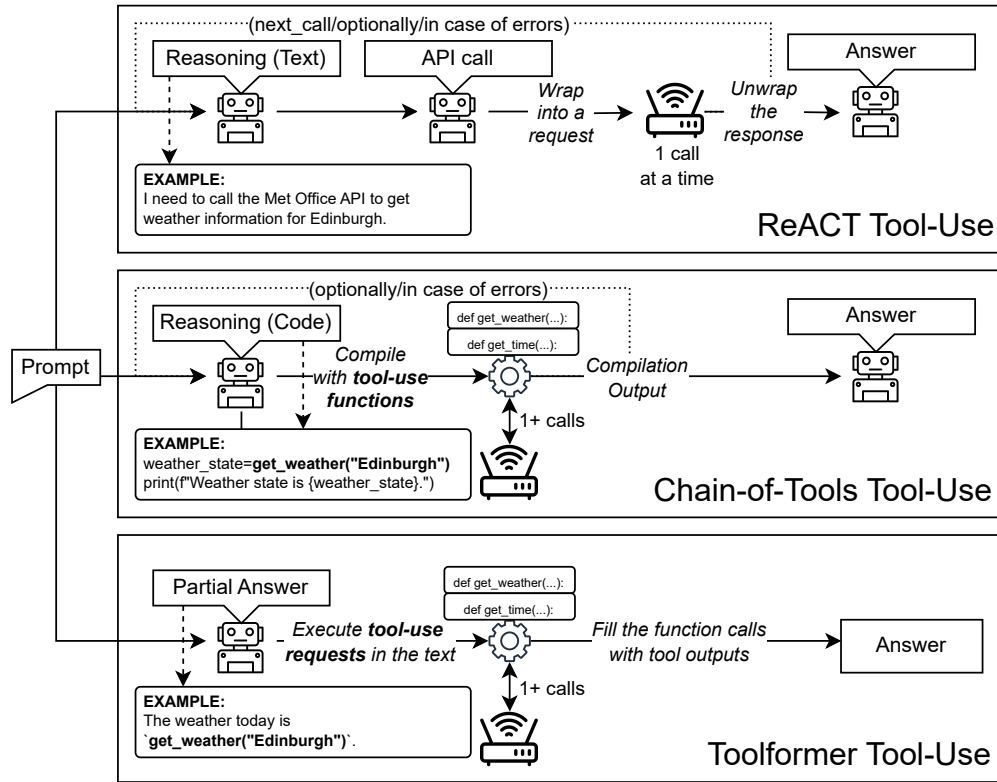


Figure 2.3: Simplified comparison of reasoning schemes: ReACT Tool-Use (Qin et al., 2023), Chain-of-Tools Tool-Use (Z. Shi et al., 2024), and Toolformer Tool-Use (Schick et al., 2023).

which is used to route the timestep in the conversation to the correct expert. The remaining experts have a simpler routing scheme, the Planner follows every Caller action, while all responses from the summariser conclude the turn. Figure 1.2 includes a simplified illustration comparing the two approaches and due to a large number of models considered at this point, a glossary was included in Figure A.1 in the Appendix.

2.2.3 Progressive Fine-Tuning

Comprehension of the overall conversation and task is crucial to every step in the pipeline; therefore, the authors of the Multi-Agent scheme argue that simple fine-tuning right from the base models might create models that are overly specialised in performing their assigned task, therefore lacking comprehension of the overall task. This prompted the authors to propose Global-to-Local Progressive Fine-Tuning (GLPFT). This scheme approaches the issue mentioned earlier by first fine-tuning a ‘backbone’ model on the entire task, where the single agent has to perform all three roles in the reasoning, and then using this model to fine-tune three models to take on specific roles. This approach

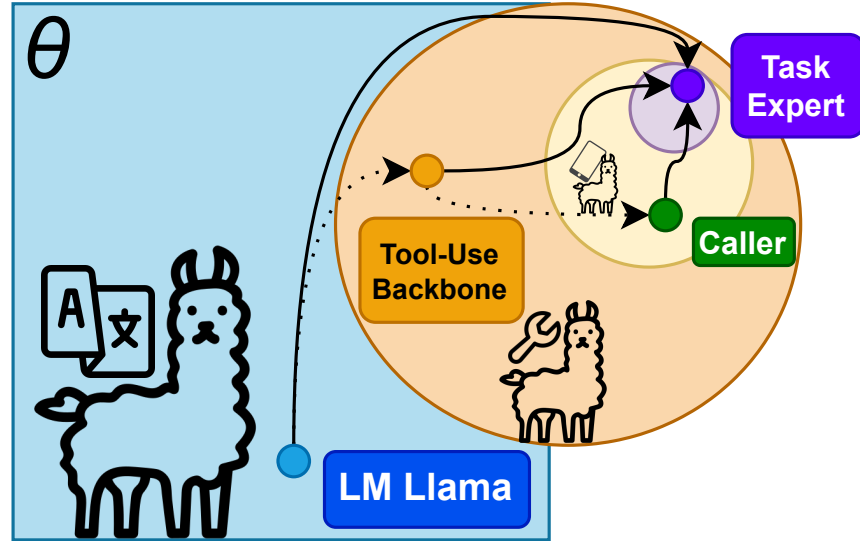


Figure 2.4: Abstract illustration of the parameter space and various approaches to training stages. The training stage represented by the dotted arrow from LM (Language Modelling) Llama to Tool-Use Backbone was first proposed by Qin et al., 2023, while the dotted line from Tool-Use Backbone to Caller represents a training regime proposed by Shen et al., 2024. The other lines depict the approaches explored in Subsection 3.3.1.

aligns with the conclusion presented by Gururangan et al., 2020, who show that a gradual multi-step alignment to the target task yields better performance than direct fine-tuning directed at the task. In this way, GLPFT indulges in applying the transfer learning principle twice through the aforementioned Global-to-Local Progressive Fine-Tuning scheme, as the first ‘Global’ stage is applied on a Llama model already pre-trained on Language Modeling. Figure 2.4 illustrates the parameter space relation between those models.

This research will expand the Multi-Agent pipeline by fine-tuning adapters applied to the Caller model, which are concerned with smaller and more specific tasks. The hierarchical structure of the problems considered allows for stacking and fine-tuning LoRA adapters that are increasingly more specialised, inspired by the stacked adapter approach used by Gema et al., 2024 in the clinical domain. I hypothesise that the Hierarchical Progressive Adapter Stacking approach will allow the resulting model to outperform the other model by incorporating knowledge from the higher domain into the lower domain, harnessing LoRAs characteristic of forgetting less of the original expertise (Biderman et al., 2024).

Chapter 3

Methodology

The methodology of training experts has been primarily adapted from the Shen et al., 2024 study. The main differences are the implementation of LoRA (Hu et al., 2021) for fine-tuning and only saving the adapter’s state after the training concludes rather than a whole merged model. The code base made public as a part of this study allows for an easy generation of training scripts that can train expert adapters. All a user has to specify to use the code is the expert they want to train and the base model. The code in the repository provides the ability to stack LoRA adapters together during training, as long as the previous adapter has already been trained. The code includes more detailed documentation about usage. Most hyper-parameters have been consistent with how the Caller was trained in the study from Shen et al., 2024. Although a more capable new iteration of this model family was released, namely, Llama-3 (L. Team, 2024), this study will maintain the use of the older Llama-2 (Touvron et al., 2023) for consistency. The length of training was fixed at the minimal number of epochs¹ where at least 1250 batches² are seen during training. One expert takes roughly 4 hours to train on a single NVIDIA A100 GPU. Additionally, it has to be noted that each adapter set after training with the current configuration occupies only about 60 MB of disk space.

3.1 Training Dataset

3.1.1 Source and Method of Generation

Training data used in this study is the ToolBench dataset, which was a part of the Toollama study (Qin et al., 2023). The dataset was generated using a zero-shot setting

¹Epoch is one entire training pass through the training data.

²The effective batch size in the study is eight samples per batch.

using GPT-3.5 Turbo (Brown et al., 2020). This automated scheme reduces the cost of acquiring this data but introduces several issues. The scheme used the RapidAPI hub, with a database of digested documentation for each tool generated by the model. Afterwards, the model is tasked with generating simulated user questions and is doing so by being instructed to create natural language queries that will use at least 2 APIs from a small subset of the APIs that were sampled from the Hub (illustrated in Figure 2.1 as Synthetic Prompt Generation). Those queries and the sampled tools were passed to GPT-3.5 Turbo in a zero-shot fashion, meaning only the Tool-Use task was explained to the model, with no examples provided. Single-Agent sub-figure in Figure 1.2 shows how the model answered queries through iterative generation.

This approach benefits from eliminating the costs of human annotation and replacing it with a more affordable OpenAI API-based solution. New tools or shifts in usage patterns can be easily accommodated by generating additional samples at any time with minimal cost overhead. This flexibility allows for rapidly producing diverse datasets that reflect current trends and requirements, enhancing the resulting models. This unsupervised generation can result in many mistakes and inconsistencies in the dataset, hence two significant issues are discussed later in this chapter. Before discussing issues with automatic generation, it's essential to acknowledge that human annotation is not perfect and prone to mistakes. An extensive research field looks into the implication and mitigation of those issues (Chiang & Lee, 2022; Pandey et al., 2022).

3.1.2 Expert-Wise Separation

Each expert is trained on a subset of tool calls that are relevant to the it. This way, the agent can focus and derive expertise in their specific task. Therefore, the dataset was split based on the tool used by the Caller in each instance on an endpoint, API family, and category level.

This split points to the fact that, in the described pipeline, every classifiable plan can be answered by one of three models: the specific endpoint trained, the API family, or the category-wide expert. In cases where no expert is available or where experts are not helpful, backing off to the base caller model is also possible.

This hierarchy poses the question of to what extent drilling down to finer tasks helps. In principle, fine task divisions should help the model focus more on the specific tasks, but acquiring experts for all endpoints is infeasible and may lead to overfitting. Therefore, balancing those two considerations requires comparing training effectiveness

Tool	Type	Acronym	Tool	Type	Acronym
Social	Category		Music	Category	
<i>chuck_norris</i>	API family		<i>deezer</i>	API family	
<i>jokes_random</i>	Endpoint	JCN_R	<i>search</i>	Endpoint	DZ_S
<i>jokes_search</i>	Endpoint	JCN_S	<i>playlist</i>	Endpoint	DZ_P
<i>jokes_categories</i>	Endpoint	JCN_C	<i>genre</i>	Endpoint	DZ_G
Education	Category		<i>infos</i>	Endpoint	DZ_I
<i>numbers</i>	API family		<i>editorial</i>	Endpoint	DZ_E
<i>get_random_fact</i>	Endpoint	N_RF	<i>artist</i>	Endpoint	DZ_A
<i>get_trivia_fact</i>	Endpoint	N_TF	<i>track</i>	Endpoint	DZ_T
<i>get_math_fact</i>	Endpoint	N_MF	<i>radio</i>	Endpoint	DZ_R
<i>get_date_fact</i>	Endpoint	N_DF	<i>album</i>	Endpoint	DZ_AL
<i>get_year_fact</i>	Endpoint	N_YF	<i>comment</i>	Endpoint	DZ_C

Table 3.1: Tools selected for experiments on hierarchy, focusing on effectiveness at various levels and the aggregation of experts into higher-order experts. Endpoint names have been shortened by omitting the API family name.

at those various levels.

The effect of the hierarchical abstraction level will be tested with an experiment where the endpoint experts and the other possible higher-order experts will be trained and evaluated. The endpoints and their corresponding Categories and APIs trained for this experiment can be seen in Table 3.1.

3.1.3 Samples Distribution Distribution

The analysis of the dataset undertaken in this study revealed that the distribution in terms of tools is not uniform nor difficulty-based. To illustrate this point, Table 3.2 contains the Top 10 most frequently used tools. Most of these tools only require 0 or 1 arguments, suggesting those are among the simpler APIs. Further, the fact that those ten endpoints alone represent over 3% (11795 of 361038) of samples from a set containing 12377 tools ($\sim 0.08\%$ of the tools), shows the significant imbalance in the distribution. The median of training samples for each endpoint is just 17.

Figure 3.1 shows that category distribution is also highly unbalanced. Over 63% of categories (31 out of 49) occupy only 25.1% of the overall dataset in aggregate. Those categories might represent the tendency of the dataset to have large blind spots for specific categories or API families. This is especially concerning given that the test set was generated similarly. Those issues are also present in that set.

I hypothesise that this discrepancy is caused by the fact that the data generation

Tool	Samples	No. Args
jokes_random_for_chuck_norris	2362	0/1
search_for_soundcloud	1510	2/2
search_for_deezer	1239	1/1
api_nutrition_data_for_edamam_nutrition_analysis	1180	0/2
random_for_manatee_jokes	1070	0/0
exchange_for_currency_exchange	1006	2/3
getmsgs_for_messages	904	1/1
jokes_search_for_chuck_norris	869	1/1
all_for_getguidelines	864	0/0
get_random_fact_for_numbers	791	1/5

Table 3.2: Top 10 most frequently occurring tools in the training set, with the number of samples and the number of arguments (No. Args) **required**/defined for each tool.

model is tasked with generating the query. In the generation, the LLM is likely to favour queries that are more natural or related to questions it saw in the training data rather than trying to maintain a good balance of samples for each API. This hypothesis is consistent with the fact that the Top 10 endpoints (Table 3.2) are all related to popular tasks, such as searching for music.

The nature of APIs means they can vastly vary in terms of difficulty and number of parameters. Some API calls only require the correct route to be specified, while many others necessitate intricate specific domain knowledge to produce valid API calls. In the future, many features of the Tool-Use pipeline can likely be enriched by generating more tailored training data, for example, by targeting challenging API categories. Subsection 3.1.1 explains that under the dataset generation strategy proposed by Qin et al., 2023 procuring those samples would not be costly, but this study will not investigate this avenue, to maintain consistency with the previous work.

This characteristic points to the necessity of inspecting the relation between the number of samples and the number of tools each expert has focused on during training. The experiments conducted by Y. Wang, Mishra, et al., 2022 inspire this experiment. Various experts with different numbers of training samples and endpoints will be compared. All experts will be trained on the Caller model to make the comparison consistent. The experts considered with the dataset sizes are shown in Figure 3.2 and the number of endpoints encompassed within an expert in Figure 3.3.

The outcome of these experiments will produce recommendations on the minimal number of samples required for training an expert and partially the extent of the benefit of acquiring more data for a specific expert. Additionally, this experiment will inspect

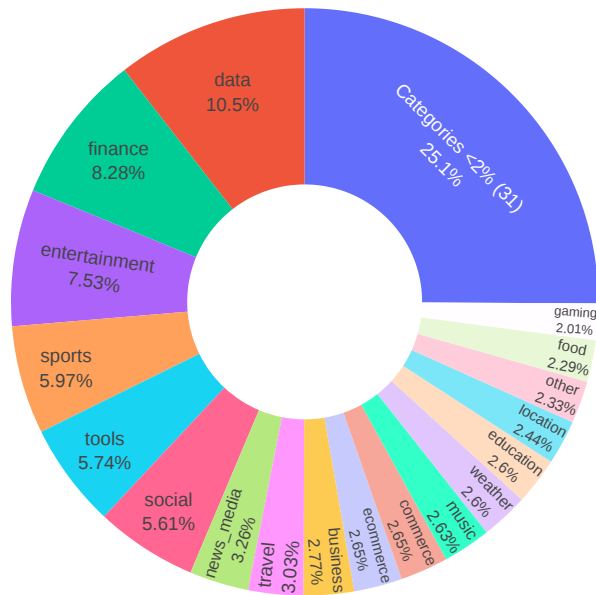


Figure 3.1: Distribution of samples by category. Exact numbers of samples are provided in Table B.1.

how larger endpoint splits may lead to diminishing returns for the expert and at what point the task splits become too coarse.

3.1.4 Classifying the Tool Request

Whenever the Planner ends the utterance with `Next: Caller`, there is an expectation in the Multi-Agent pipeline that the action immediately following is a call to a single API before the pipeline can invoke the planner again. Therefore, in the ideal world, the Planner provides a clear choice of one tool. If the tool selection is clear, the pipeline can decide which adapter is the most appropriate for this utterance and load it to improve the output quality. When a plan is generated without a clear indication of the tool, all expert adapters can be disabled as back-off to handle these cases, though this means no performance improvements can be harnessed. Figure 3.4 illustrates the importance of the tool classifier in the step between the Planner and the Caller.

The scheme used for identifying which tool was requested by the Planner is a simple rule-based system, presented in Algorithm 1. At its core, this algorithm performs simple string matching with the list of retrieved tool names. It is a simple yet effective baseline. Still, the inspection of the dataset quickly revealed that this scheme gets confused by cases where the Planner comments on the output of the previous API, sometimes with strong wording such as `the function X is not a valid approach` or with weaker wording `the function X has been tried in the plan`. In those ‘negation’

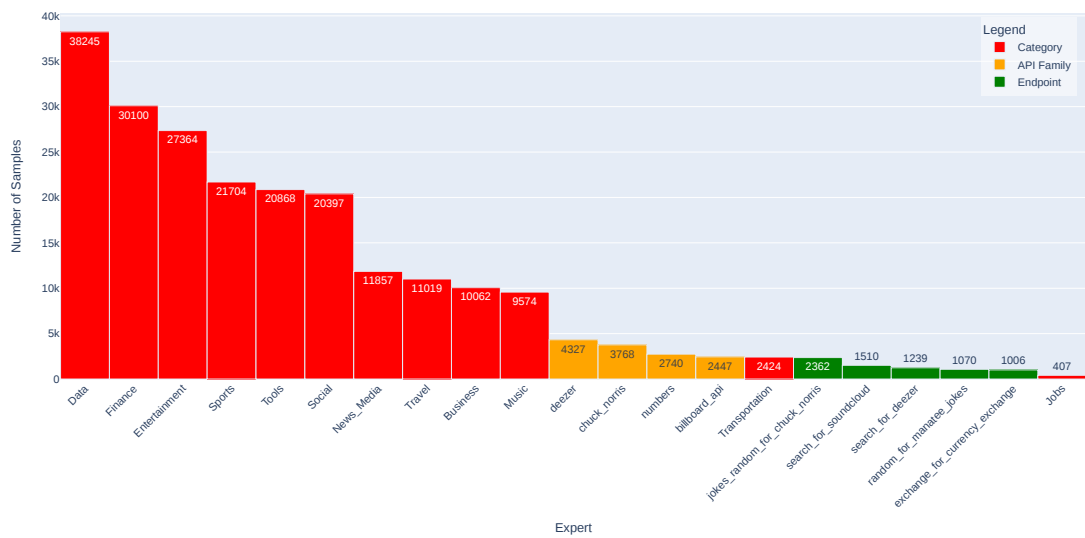


Figure 3.2: Number of samples per tool considered for comparing the impact of dataset size. Detailed information is also available in Table B.4 in the Appendix.

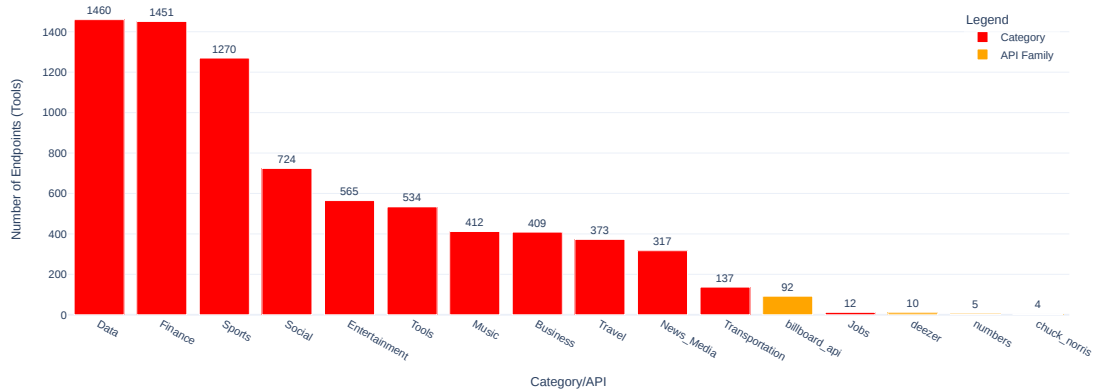


Figure 3.3: Number of tasks encompassed by each category/API family. Detailed information is available in Table B.4 in the Appendix.

cases, the simple approach will incorrectly believe that the Planner requested a function that it, in reality, was only commenting about. The distinction between strong and weak wording is significant as weak negation wording is often used in cases of reattempting a specific tool (for instance the function X has failed, but it will be tried again by modifying...).

Typical instances of ‘negation’ wording have been identified in the training set semi-automatedly by classifying using the simple scheme and recording all cases with more than one tool mentioned in the plan. This set of ambiguous instances is then used to find the most common contexts before and after mentioning each API and the most common phrases used around those. That has produced hundreds of potential candidates that have been manually inspected to find phrases indicating negations, classifying them

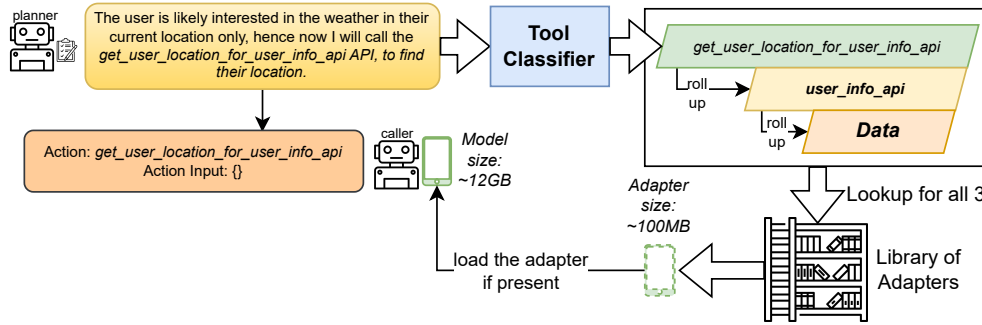


Figure 3.4: Illustration of how the plan determines the adapter choice in the pipeline, with a focus on the tool classifier.

into strong and weak negations. Afterwards, this expanded scheme was compared with the original classification scheme.

Figure 3.5 shows the difference in terms of the ability to parse (over 15% reduction in the cases of inability to classify) and the accuracy of the resulting classification (nearly 5% reduction in miss-classifications distribution of the parsed set). Notably, the raw number of miss-classifications has risen because the improved scheme is more likely to parse in general; therefore, it is more likely to parse incorrectly. It's worth highlighting that the probability of it parsing incorrectly, given that the sample was parsed, was reduced (as shown on the smaller charts in Figure 3.5), which is a more important outcome.

An alternative approach is to train a new Planner that is more decisive and provides a more explicit structure for indicating API choices. This would involve modifying the dataset, which, though costly, could be done automatically using a large language model to rephrase the plans to better suit the subsequent calls by retrospectively inspecting the entire conversation. This will enable the model to ensure consistency like it was not able to during the original generation because of the word-by-word nature of decoding, which does not allow for mistakes or inconsistencies to be fixed.

This study will not investigate this approach as it would make distinguishing between the benefits of being more decisive and the benefits of the Expert pipeline more difficult to track. Hence, the scheme used is a simpler rule-based system explained previously. The 'decisiveness' approach is reserved as a potential avenue for future work.

3.1.5 Certainty Separation

The dataset resulting from the separation discussed in subsection 3.1.2 has a significant issue. During inference, the model can only rely on the tool classifier to load the correct



Figure 3.5: Comparison of tool classification schemes, considering both the entire dataset (upper row) and the distribution of mistakes within the subset of the dataset that could be parsed (lower row).

expert patch into memory, as there is no ground truth for which API is the correct follow-up available. Therefore, the expert will never be loaded in cases where the plan does not allow the classifier to make a classification, and the model will rely on the general model to generate the API call by turning off all adapters. Hence, the expert adapter will never see ‘uncertain’ plans at inference time due to the nature of the inference pipeline. For a plan in the training set to be deemed as ‘uncertain’ if they failed to mention a tool, mention multiple tools or the tool mentioned does not correspond with the tool used.

Y. Wang, Mishra, et al., 2022 show that in NLG (Natural Language Generation), the number of samples has a diminishing effect on the quality of the resulting model, pointing to the belief that reduction in train set size due to this removal should have a limited adverse impact. However, it is crucial to assess this relation in this specific task. Therefore, the effect of excluding and including ‘uncertain’ samples will be analysed. Considering 15 distinct tools, the resulting performance of training on the following two datasets will be compared. The first dataset is all samples where the Caller produced an API call for the tool targeted (called *ALL*), whilst the second *CERTAIN* set contains samples from the *ALL*, but cases where tool classification was not possible and cases where the classification did not match the subsequent API call were filtered

Algorithm 1 Classify Plan Into a Single Tool

```

1: function REMOVENEGATIONS(plan, negations)
2:   for each negation in negations do
3:     plan  $\leftarrow$  plan.REPLACE(negation, "[negated statement]")
4:   end for
5:   return plan
6: end function

7: function FINDCOMMONSTRINGS(plan, tool_names)
8:   common_strings  $\leftarrow$  []
9:   for each tool in tool_names do
10:    if tool is in plan then
11:      APPEND(common_strings, tool)
12:    end if
13:  end for
14:  return common_strings
15: end function

16: function CLASSIFYPLAN(plan, tool_names, strong_negations, weak_negations)
17:   plan  $\leftarrow$  REMOVENEGATIONS(plan, strong_negations)
18:   common_strings  $\leftarrow$  FINDCOMMONSTRINGS(plan, tool_names)
19:   if LENGTH(common_strings) = 1 then
20:     return common_strings[0]
21:   else if LENGTH(common_strings) > 1 then
22:     plan  $\leftarrow$  REMOVENEGATIONS(plan, weak_negations)
23:     common_strings  $\leftarrow$  FINDCOMMONSTRINGS(plan, tool_names)
24:     if LENGTH(common_strings) = 1 then
25:       return common_strings[0]
26:     end if
27:   end if
28:   return None ▷ Cannot be parsed
29: end function

```

out. Examples of cases that would be filtered out to get the *CERTAIN* set are shown in Figure 3.6.

The selection of tools was made to provide a variety of dataset sizes and various levels of prevalence of unclassifiable samples, as shown in Table 3.3. The size of the training dataset ranged from 229 to 38245, and depending on the tool, the proportion of unclassifiable cases ranged from 33.2 % to 57.9 %. The base model used for each experiment is the Caller model trained by Shen et al., 2024.

CONFUSING Planner: Based on the user’s request for recent hashtags in Argentina, I should call the **get_country_s_hashtags_for_twitter_hashtags** function with the argument "country: Argentina". (...) I need to make another API call to fetch all recent hashtags worldwide, as the user expressed interest in them as well. This additional call will provide a comprehensive list of global trending hashtags.

Reference Caller: **get_worldwide_hashtags_for_twitter_hashtags**

Tool Classifier: **get_country_s_hashtags_for_twitter_hashtags**

AMBIGUOUS Planner: 1. Use the **search_for_deezer** function to search for popular tracks in the pop music genre. 2. Use the **exchange_for_currency_exchange** function to get the currency exchange rate between British Pounds and Euros. 3. Use the **album_for_deezer** function to get the details of the latest album by Taylor Swift. I will start by using the **search_for_deezer** function to search for popular tracks in the pop music genre.

Reference Caller: **search_for_deezer**

Tool Classifier: **[AMBIGUOUS]**

UNEXPRESSED Planner: The previous action of fetching the current stock information for GOOG on NASDAQ failed with a "Not Found" message. Since this action did not provide any useful information, I will try a different approach to download the portfolio of freefloat investments.

Reference Caller: **portfolio_for_freefloatus**

Tool Classifier: **[NO TOOLS MENTIONED]**

Figure 3.6: Instances from the training set where the plan fails to lead to the correct tool classification according to the scheme used in this study, with the referenced endpoint shown afterwards. The word underlined in the first example is the most likely cause of confusion.

3.2 Test Sets

3.2.1 ToolBench Test Set

The major conclusions of the viability of this pipeline will be drawn based on the ToolBench test set, as it has APIs covered by the experts trained as part of this study. This will also allow for a good comparison with the gains from the Multi-Agent pipeline proposed by Shen et al., 2024. Furthermore, as this set is the main focus of this study, two models from the OpenAI’s GPT family will be compared against the proposed expert model: GPT-4o mini and GPT-3.5 Turbo. The latter will be used in both zero-shot and few-shot (5 samples) in-context learning strategy, whilst GPT-4o mini will be only tested in a zero-shot setting (due to usage constraints).

Tool	Training samples		Diff (%)	Acronym
	<i>ALL</i>	<i>CERTAIN</i>		
Data (<i>Category</i>)	38245	21356	44.3 %	D
chuck_norris (<i>API family</i>)	3768	2138	43.4 %	CN
billboard_api (<i>API family</i>)	2447	1407	42.5 %	BA
jokes_random_for_chuck_norris	2362	1314	44.3 %	JCN
Transportation (<i>Category</i>)	2424	1431	40.9 %	TR
search_for_soundcloud	1510	717	52.6 %	SS
search_for_deezer	1239	520	57.9 %	SD
random_for_manatee_jokes	1070	714	33.2 %	RMJ
exchange_for_currency_exchange	1006	505	49.8 %	ECE
get_year_fact_for_numbers	372	229	38.3 %	GYF

Table 3.3: Comparison of training set sizes before and after removal.

3.2.2 ToolAlpaca Test Set and Additional Annotation

Assessing the out-of-domain capabilities of the pipeline proposed necessitated using a dataset with tools unknown to the experts and Caller, which in this case will be the test set from the ToolAlpaca study (Tang et al., 2023). As none of the tools from this test set are previously known, only category-wide experts can be utilised. However, the categories of each tool are also not known, hence all the tools with the associated documentation were compiled into the prompts for GPT-4 Turbo, which is tasked with annotating each API with one of the 49 categories from RapidAPI Hub. This way, 297 APIs were annotated, with only 11 cases of hallucination where the model has made up a new category. The hallucination cases were annotated manually. The prompt template with an example is included in the Appendix C.1. This test set was also used to assess the Multi-Agent approach presented by Shen et al., 2024.

3.2.3 Specific Developmental Test Set

In most tools, the original ToolBench (Qin et al., 2023) test set has an insufficient amount of cases, making comparisons between specific adapter approaches difficult. Therefore, the original training set was split into two sets to make better-guided design decisions: a new training set (90%) and a new specific test set (10%). This ratio was kept consistent for all tools to ensure that there was a sufficient number of test cases for any tool. This new test set will only be used to make comparisons between approaches to procuring adapters and will not be considered when considering the overall impact of the pipeline. Due to this separation, all of the experts in this study will be trained on

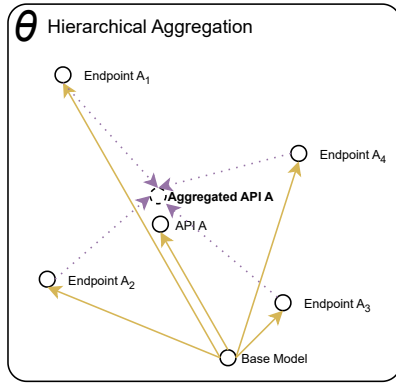


Figure 3.7: Illustration of the parameter-space effect of aggregating experts.

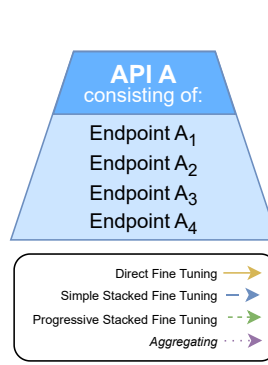


Figure 3.8: Illustration of the parameter-space effect of stacking adapters in both hierarchical and simple approaches.

slightly smaller training sets.

3.3 Training Scheme Experiments

3.3.1 Selection of the Base Model

Every expert has to be based on some pre-trained model. The most traditional approach uses a base foundation model, in this case, the base Llama-2 7B model (Touvron et al., 2023). This is the selection that Qin et al., 2023 have made for their model, but based on the findings of Shen et al., 2024, the model is likely to benefit from a training regime similar to GLPFT, where the Backbone model will be used as a base to allow Tool-Use knowledge of the Backbone to be used by the Expert model. This scheme is how the Caller model was trained (with the difference of using full fine-tuning rather than PEFT), and this points to the third possible base model selection, which is to utilise the Caller model, hence creating a training scheme that one may call Global-to-Local-to-Expert Fine-Tuning due to its three distinct fine-tuning stages. The three possible approaches have been shown in Figure 2.4. The tool choice is the same as in Table 3.3; each tool was trained on all three base models.

3.3.2 Aggregating Experts into Higher-Order Experts

One of the hypotheses this research attempts to test is that lower-order task experts can be aggregated into new higher-order experts without additional training, as illustrated in Figure 3.7. Prior research has used simple aggregation schemes to combine models

trained on arbitrary task divisions into models that outperform direct training on the task (Y. Wang, Agarwal, et al., 2022). For this experiment, the endpoint experts trained for the experiment described in Subsection 3.1.2 were aggregated and compared against their corresponding directly trained higher-order experts.

Suppose models acquired in this manner match the performance or perhaps outperform the corresponding directly trained model. In that case, this training approach can be advantageous for tool sets that frequently change or expand. In those cases, a change in one endpoint will only require retraining on that endpoint, followed by an aggregation with the remainder of the set. Due to its straightforwardness and reproducibility, the aggregation scheme used is the arithmetic mean. The focus of this experiment is concerned with the effect of aggregation in general rather than narrowing it down to a specific aggregation scheme, hence the preference for this scheme.

3.3.3 Hierarchical Progressive Adapter Stacking

Target Task	Adapter 1 Task	Adapter 2 Task	Adapter 3 Task	ACR.
jokes_random_for_chuck_norris	Social <i>Target</i>	chuck_norris <i>Target</i>	<i>Target</i> <i>Target</i>	JCN_R
jokes_search_for_chuck_norris	Social <i>Target</i>	chuck_norris <i>Target</i>	<i>Target</i> <i>Target</i>	JCN_S
jokes_categories_for_chuck_norris	Social <i>Target</i>	chuck_norris <i>Target</i>	<i>Target</i> <i>Target</i>	JCN_C
chuck_norris	Social <i>Target</i>	<i>Target</i> <i>Target</i>		CN
free_nba	Sports <i>Target</i>	<i>Target</i> <i>Target</i>		FN
currency_exchange	Finance <i>Target</i>	<i>Target</i> <i>Target</i>		CE

Table 3.4: Tasks considered for adapter stacking, with explanations for both the progressive (first row of each target task) and simple stacked approaches (second row).

Prior research (Gema et al., 2024; Lialin et al., 2023) has used stacking of LoRA (Hu et al., 2021) adapters to allow the model to achieve performance beyond what is achievable with one adapter. The parameter space interpretation is shown in Figure 3.8. An approach based on this idea will be investigated in two experiments, one where the task used for all stages was the same target task (inspired by pre-training used by Lialin et al., 2023), and where progressively, each adapter focuses on a more specific task

(effectively drilling down, similarly to how Gema et al., 2024 used an initial adapter to bring the model into the domain of the target task before focusing on the task). Table 3.4 presents the experts trained for this experiment.

Chapter 4

Results

The resulting experts from the experiments outlined in the previous chapter are evaluated against reference API calls from ToolBench, ToolAlpaca and the custom-specific test set. The metrics considered are the exact match score assessing the correctness of the tool selection and the F1-score¹ over the correctness of the arguments the API call gets populated with (both the keys and values provided). In the spirit of focusing on the adapter part of the pipeline, the results will be grouped based on the output of the Tool Classification. The cases where the pipeline was unable to classify will not be shown, as they will simply not be impacted by the expert adapter approach at all, with them being run by the base Caller model. The results show that applying the adapters has a positive impact on the accuracy of the API Calls produced by the pipeline and enables the extraction of heuristics that guide the optimal training of experts.

4.1 Optimal Hierarchical Level of Abstraction

The investigation results into the correct abstraction are presented in Figure 4.1. They reveal that the category experts struggle to offer considerable improvements over the Caller from Shen et al., 2024 baseline. On the other hand, both the API family-wide and endpoint-specific models were closely matched in terms of performance, and both were significantly better than the baseline. This outcome points to a preference for training API family-wide experts, those adapter experts can handle multiple kinds of tool invocations to a satisfactory level, reducing the expense of training multiple adapters for a comparable effect. Interestingly invocations of `search_for_deezer` (DZ_S) were handled significantly better by the API-family expert, rather than the

¹F1-score is a correctness metric that balances precision and recall. $F1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

endpoint expert, which seems counterintuitive. However, what likely is causing this effect is the generalizability of information contained in other endpoints of the deezer API family that have helped with that specific endpoint.

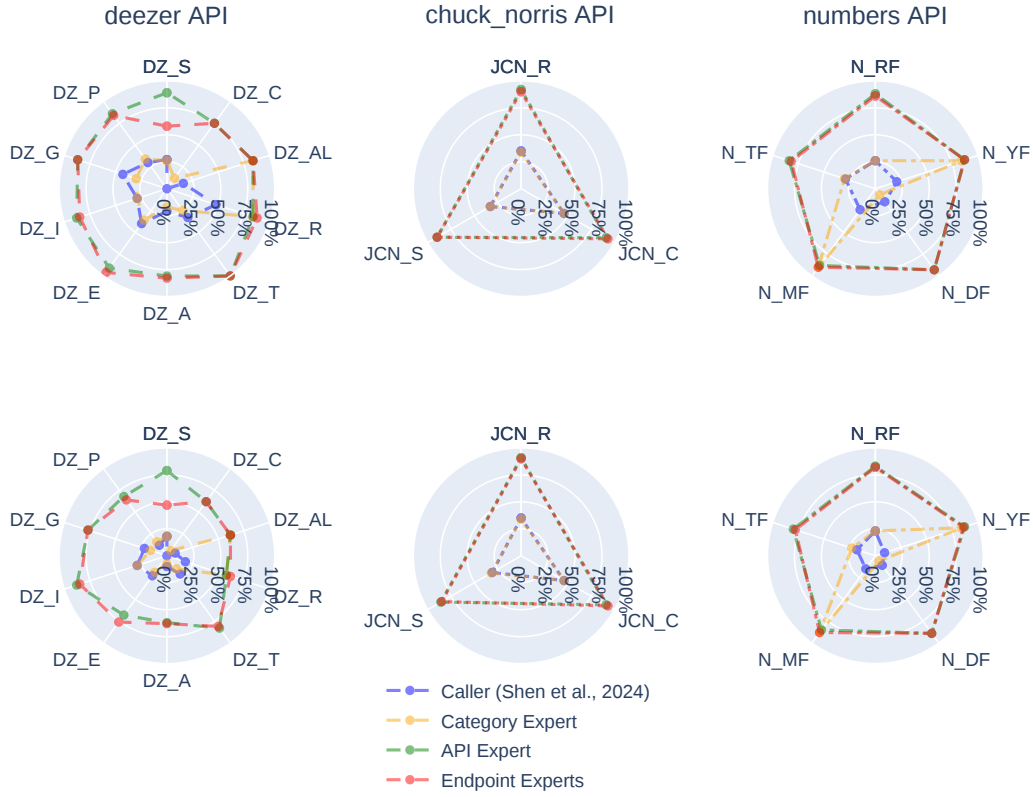


Figure 4.1: Comparison between hierarchical experts and the baseline for each task. The upper row shows Exact Match performance on API choice, while the lower row shows F1 scores for Argument Population. Exact performance figures are listed in Table B.3 in the Appendix, and tool acronyms are disambiguated in Table 3.1.

4.2 Effect of Sample Number and the Tasks Considered

Figure 4.2 shows how the performance of the experts evolves as more tasks (endpoints) are considered and as the number of samples grows. The plot presents two intuitive trends, firstly below 1000 samples, the expectation of the performance cannot be too high, as the expert likely overfits the samples provided. Secondly, as the model’s focus broadens, the gain over the baseline Caller model shrinks, ultimately becoming practically non-existent past 200 endpoints per expert. This observation is consistent with the expectations because, in the limit of tasks (i.e. the entire training set), the expert is trained on the same data as the Caller model. One unintuitive trend is that

the increase in the number of samples seems to harm the resulting model. Still, I hypothesise this is simply the fact that experts with more samples also had more tasks, hence the tendency in the samples graph is explainable through the previous conclusion. The main conclusion from this experiment is that experts should be trained for sets of endpoints below 200 to maximise the resulting performance. Regarding the number of samples, the experiments suggest procuring a dataset of over 1200 samples for each expert.

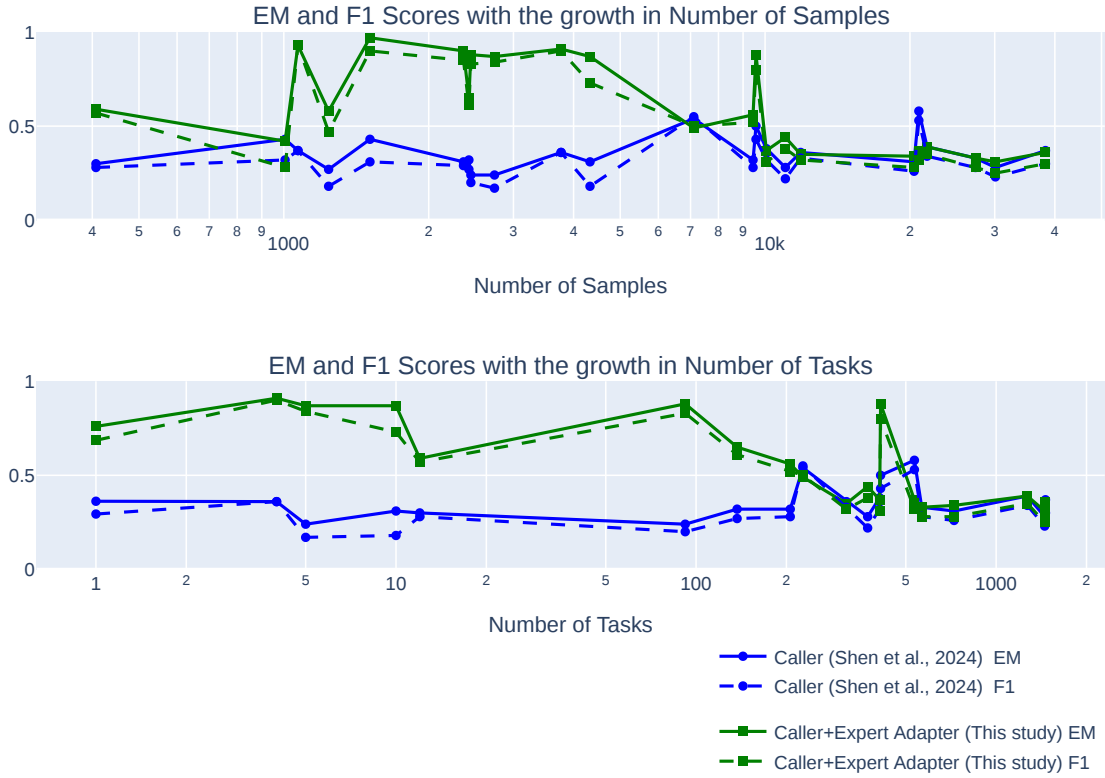


Figure 4.2: The effect of increasing the number of tasks and samples on pipeline performance compared to the baseline. Exact metrics are provided in Table B.4 in the Appendix.

4.3 Effect of Limiting the Dataset to Certain Plans

The hypothesised harm of the uncertain plans has been proved to be incorrect to an extent, as, in practice, the model has benefited from the additional cases and the noise provided by those cases. This effect can be seen in Figure 4.3. Certain plans may have repetitive patterns that the model started to overfit without uncertain plans or simply the number of samples was insufficient after the filtering. Therefore, all training samples

have been used for the remainder of the experiments, and it is recommended that the proposed removal is not performed due to its harmful effects.

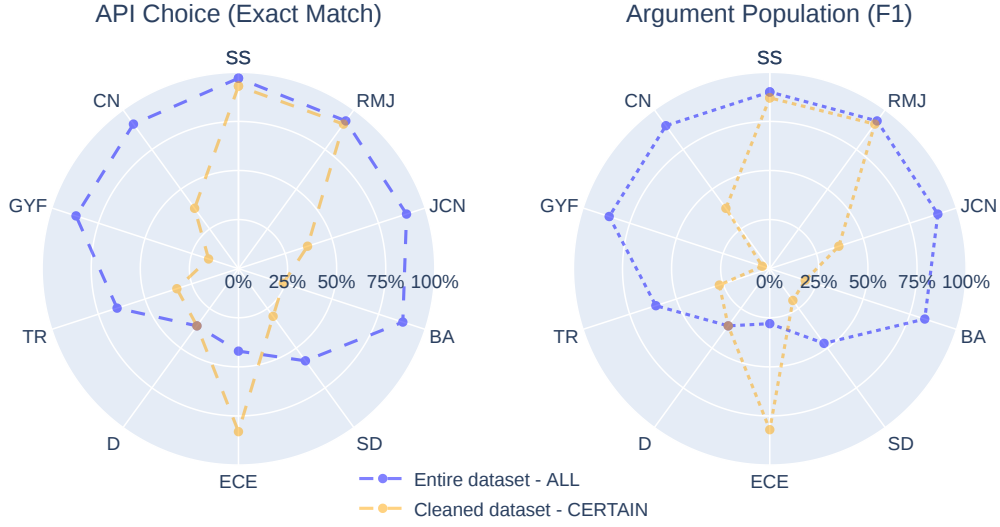


Figure 4.3: Expert performance considering both the entire training set and a subset limited to specific samples. Task acronyms are disambiguated in Table B.5. The data used for this plot is also provided in Table B.6 in the Appendix.

4.4 Preferred Base Model Selection

Qin et al., 2023 propose fine-tuning a Tool-Use wide model (called Backbone in the experiments) to gain performance over the standard Llama model (Llama) (Touvron et al., 2023) on this task, and Shen et al., 2024 has made this pipeline more complex by separating them into multiple agents, amongst them a model dedicated for writing API Calls (Caller). Each of the three models described, Llama, Backbone, and Caller, can serve as the basis for the expert model. The results in Figure 4.4 show that each step is beneficial, especially holding off on adding the expert until the Caller model is trained. In most tasks, the Expert based on the Caller model vastly outperforms any other path of acquiring a model for this task.

The only endpoint that suffered from the introduction of the expert adapter is `exchange_for_currency_exchange` (ECE) but it was also the only endpoint that was aided by the filtering in the experiment discussed in Section 4.3. Hence, the lacklustre performance can be explained by the lack of filtering or the wrong level of abstraction used. It has to be noted that applying the expert adapter to the Caller model has the highest theoretical computational expense. Still, the cost of training a Caller and then the

Expert Adapter is much lower than the previous stages of the training (training the Caller is roughly one order of magnitude shorter than the Backbone, and the expert adapter is approximately two orders of magnitude shorter). The outcome of this experiment confirms that the Multi-Agent approach proposed by Shen et al., 2024 is beneficial in terms of API Call correctness as the Caller has outperformed the Backbone in most comparisons.

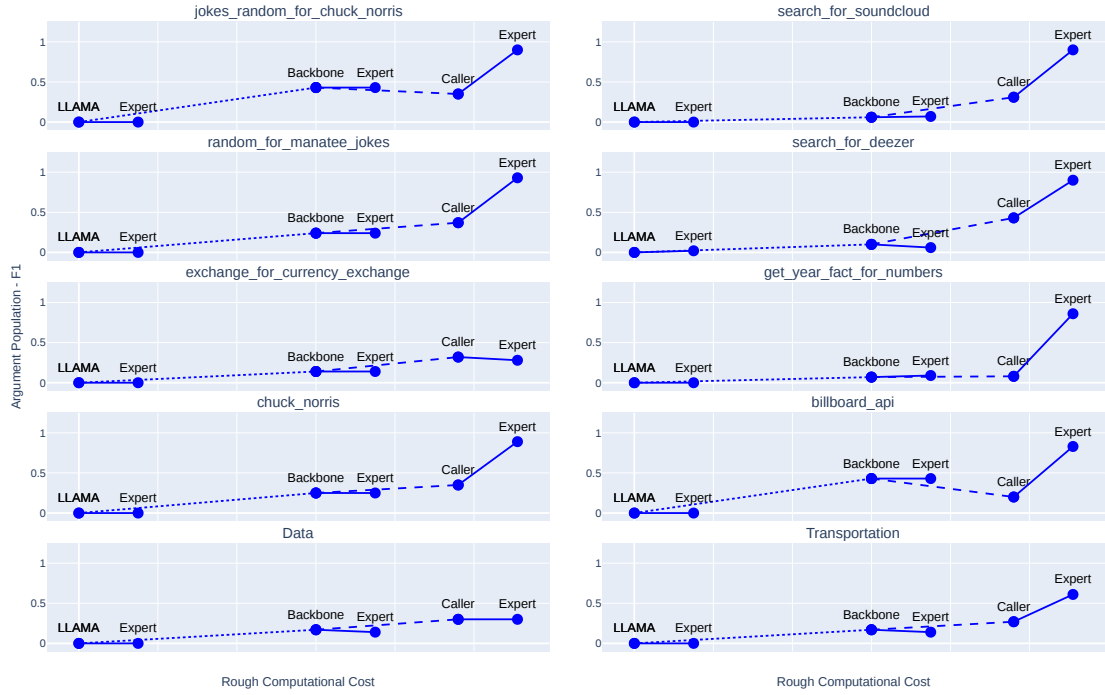


Figure 4.4: The Argument Population F1-score depending on training the expert adapter with various base models. Note that the ‘computational cost’ axis is a rough illustration rather than a precise representation. Performance metrics are listed in Table B.7.

4.5 Viability of Expert Aggregation for API Experts

Aggregating the adapters fine-tuned for the Endpoints of a particular API to get an API Expert has been shown to work in the cases considered, as seen in Figure 4.5. The Aggregated Expert has achieved results comparable to those of the directly trained API Expert in these experiments. Therefore, APIs affected by frequent changes can be reflected dynamically by training endpoint experts and aggregating them. Each change can be accommodated by training affected endpoint experts alone and then aggregating again.

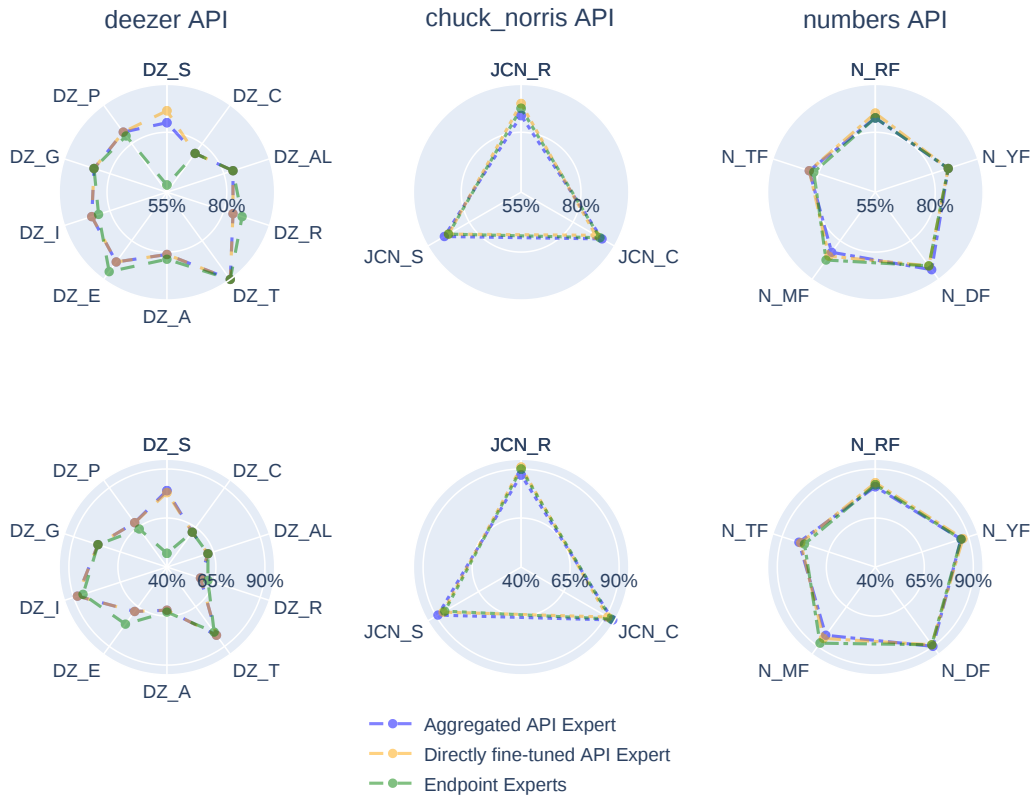


Figure 4.5: Comparison of performance between the aggregated API expert, directly trained API expert, and individual experts. The upper row shows Exact Match on API choice, while the lower row shows F1-scores for Argument Population. Acronyms are disambiguated in Table B.2, and exact figures are provided in Table B.8.

4.6 Progressive Adapter Stacking for Boosting Performance

Figure 4.6 illustrates that stacking adapters with the training scheme used is only beneficial in cases where the direct fine-tuning has failed to improve the performance. Additionally, making the stacking progressive does not make a significant difference. This may be because the base model has already undergone progressive fine-tuning phases, and the residual knowledge is plentiful for the task. However, in many cases, progressive stacking may require less training overall as the intermediate adapters can be reused. Therefore, for the Expert Tool-Use pipeline, the recommendation is to use a progressive stacked adapter approach only when the performance is disappointing.

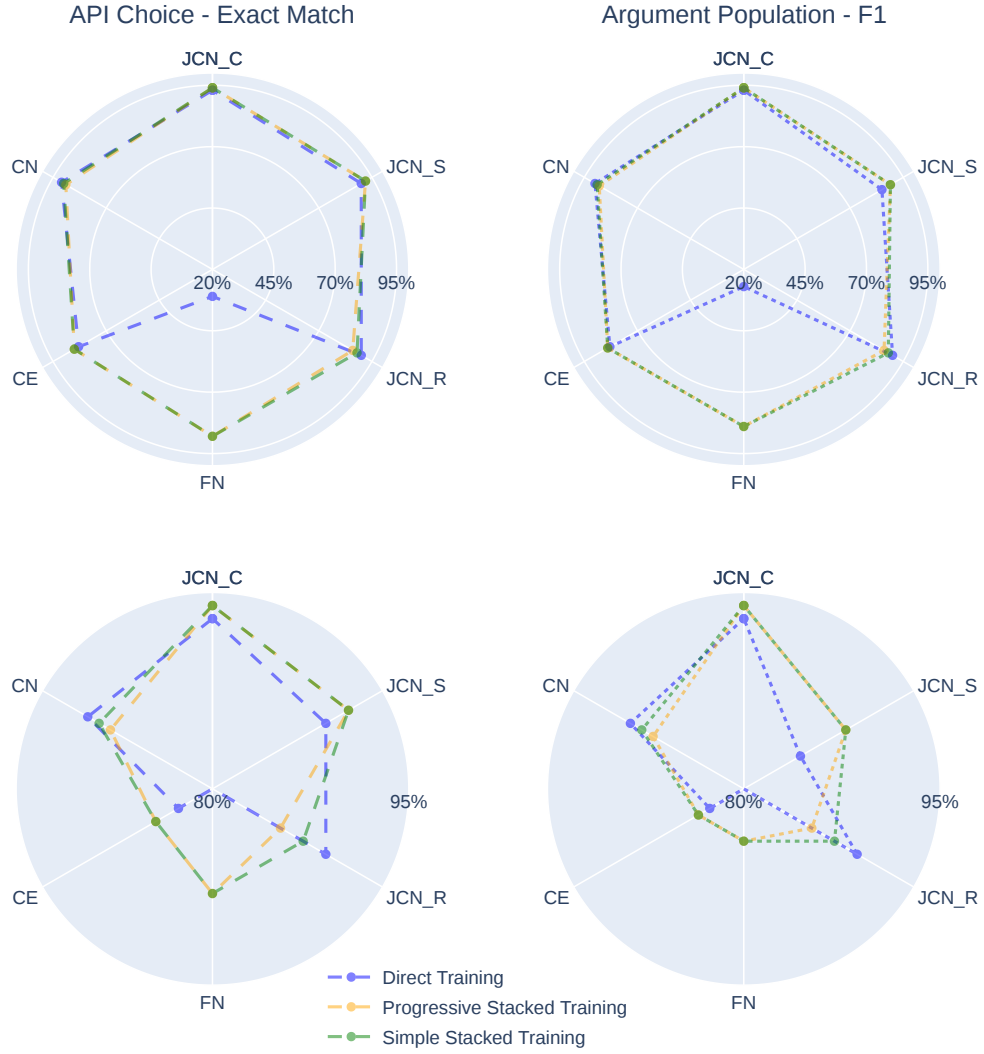


Figure 4.6: The effect of parameter stacking on performance, comparing simple stacking and progressive stacking. The lower row shows the same data zoomed into the upper range. Tasks are disambiguated in Table 3.4, and performance figures are listed in Table B.9 in the Appendix.

4.7 In-Domain and Out-Of-Domain Test and Comparison

The in-domain tool comparison compares the Expert pipeline against the Caller it is based on, and two closed-source competent models, with one of them, trialled both in zero-shot and few-shot settings. As can be seen from Figure 4.7, the proposed pipeline significantly outperforms the baseline of Caller. Still, it retains its edge over the closed-source models in most categories. This outcome is especially remarkable when considering that the ToolBench (Qin et al., 2023) dataset was generated by GPT-3.5 Turbo itself, and the Expert model is merely attempting to mimic its behaviour through distillation (Iliopoulos et al., 2022).

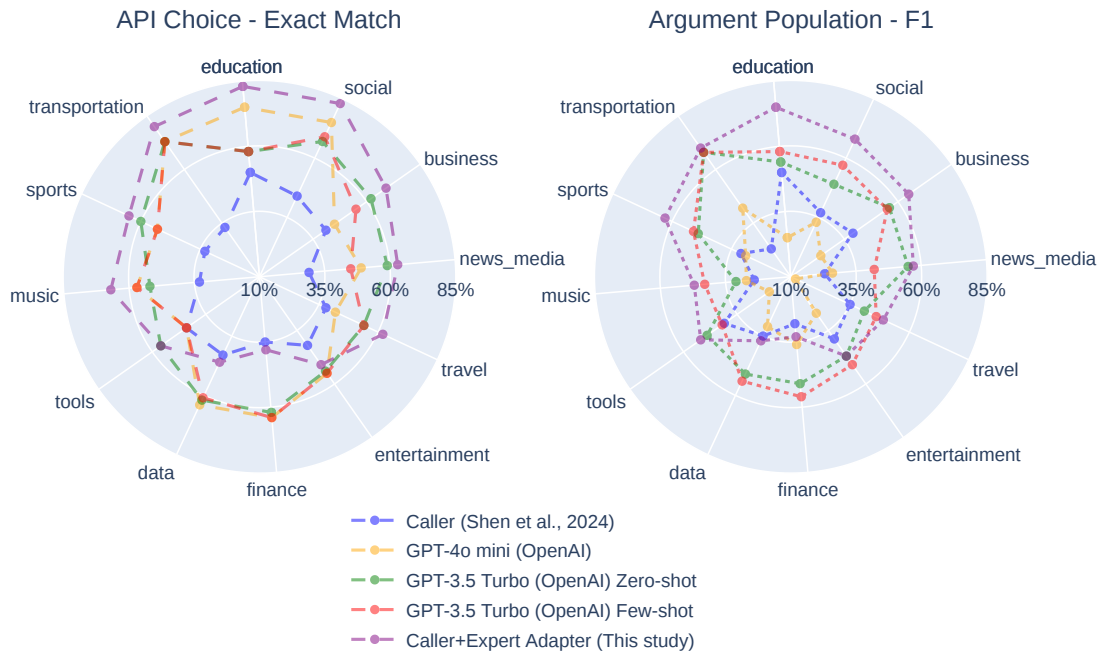


Figure 4.7: ToolBench test results for Callers, GPT models, and Experts. Metrics are also included in Table B.10.

Furthermore, in line with the findings of Hsieh et al., 2023, the few-shot inclusion of examples has not substantially helped the GPT-3.5 Turbo model, sometimes even harming the model. This result might be due to the length of the conversations, which can cause the model to get confused about that is the relevant data, similar to the distraction effect discussed by F. Shi et al., 2023. Interestingly, the newest model from this experiment, GPT-4o mini, performed the worst regarding Argument Population. Additionally, the pattern of preference for smaller task splits discussed in Section 4.2 is still present in this data. The experts trained on the two largest categories Data (1460 endpoints) and Finance (1451 endpoints) had offered minimal gains and performed worse than the closed source models, whilst the categories where the Expert model performed the best were relatively small, for instance, Education (206 endpoints) and Transportation (137 endpoints).

The final experiment uses the ToolAlpaca dataset (Tang et al., 2023) to test the Out-Of-Domain gains of this pipeline. The fact that the experts are trained to specialise more in the tools outside of this test set is entirely consistent with the test outcome, as the gains are small. Despite being numerically smaller the gains in terms of Argument Population are more insightful as they show that for the majority of the categories, the ability to choose the correct parameters has improved noticeably, but still falls far from what would be required for a viable practical solution. On aggregate, the Expert

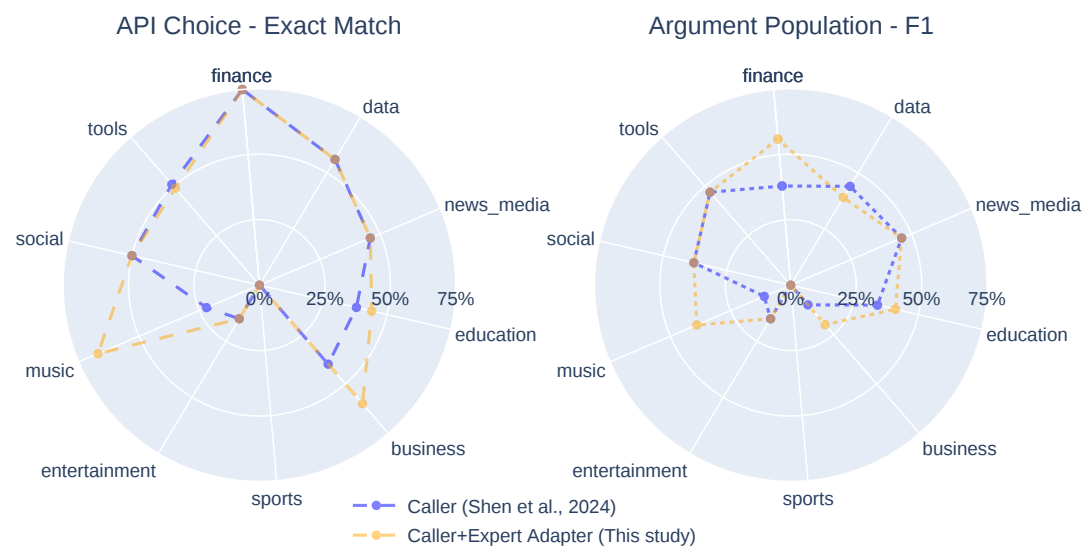


Figure 4.8: ToolAlpaca out-of-domain test results for callers, GPT models, and experts. Metrics are also included in Table B.11.

pipeline was still better than the baseline Caller model, hence pointing to the fact that the resulting experts could learn additional generalisable information about API categories that the original Caller lacked.

Chapter 5

Conclusions

5.1 Limitations and Future Work

The choice of Llama-2 as the base model may have limited the potential performance. Therefore, to fully leverage the proposed pipeline, it is essential to explore the effects of this approach using a more advanced and capable base for the Caller. The pipeline shown in this study has not investigated all endpoints, API families and categories due to computational constraints. Hence any application of this approach must be evaluated before deployment. The provided code and the new specific test set split should aid in the validation effort.

Uncertainty is a major consideration when aspiring to improve the pipeline as currently only 64.3% of the plans in the training set can be classified into a choice of tool. Additionally very often when the tool is classified the API call does not match the classification. This means in many cases no experts can be loaded due to insufficient classification abilities, and in some cases, the incorrect expert is loaded. This is the reason why API Choice Exact Match scores rarely achieve 100% when endpoint experts are used, as sometimes the expert has to deal with cases where the tool classifier has incorrectly invoked them. Hence, a more involved expert selection scheme or retraining of the Planner to have a structured way of indicating the tool required for the next step would be beneficial. Retraining the Planner requires modifying the training set to ensure that the required tool is indicated in the output sequences for the Planner and ambiguities are avoided.

The final avenue for future work is the investigation of the poor performance of GPT-4o mini during Argument Population, which was not explored as this model was used merely for comparative purposes in this study. The lower cost of GPT-4o mini

makes it a very palatable option for inference and data generation if the issue identified can be remedied. At the same time, in the current state, it is questionable whether it is worth using the newer model given the results presented.

5.2 Summary

This study proposed an expert adapter pipeline for progressing the advances made by Shen et al., 2024 in writing API calls. It was shown that further reduction in the reliance on the zero-shot In-Context learning abilities of the model has a significant positive effect on the final performance, even when compared with much larger closed-source models. The correct heuristics for implementing this pipeline in practice have been investigated, with the conclusion that experts focusing on less than 200 endpoints tend to perform substantially better than the Caller model, whilst experts with a broader focus tend to be unable to beat the baseline. Furthermore, endpoint aggregation into API family experts is a viable approach that can lower the expert pipeline's computational overhead for often modified APIs. Progressive fine-tuning was also beneficial to the entire pipeline as the best base model considered was also the model that was trained in multiple stages. Furthermore, stacked adapter approaches can help with endpoints that direct fine-tuning schemes struggle with. Still, this approach has little to no benefit if the expert is already performing well. Therefore, this study proposes that partial implementations of Tool-Use can train experts focusing on popular APIs or tools that base solutions struggle with to improve performance. This can be done using adapters that can be loaded dynamically and additionally could be trained and distributed by the API creator if desired, due to their small size.

Bibliography

- Asai, A., Zhong, Z., Chen, D., Koh, P. W., Zettlemoyer, L., Hajishirzi, H., & Yih, W.-t. (2024). Reliable, adaptable, and attributable language models with retrieval.
- Béchar, P., & Ayala, O. M. (2024). Reducing hallucination in structured outputs via retrieval-augmented generation.
- Biderman, D., Ortiz, J. G., Portes, J., Paul, M., Greengard, P., Jennings, C., King, D., Havens, S., Chiley, V., Frankle, J., Blakeney, C., & Cunningham, J. P. (2024). Lora learns less and forgets less. <https://arxiv.org/abs/2405.09673>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners.
- Canalys. (2021). Global smart speaker market set to reach 163 million units in 2021. *canalys.com*. Retrieved April 5, 2024, from https://canalys-prod-public.s3.eu-west-1.amazonaws.com/static/press_release/2020/SSPR2020Q2.pdf
- Chiang, C.-H., & Lee, H.-y. (2022). Re-examining human annotations for interpretable nlp. <https://arxiv.org/abs/2204.04580>
- Curran, S., Lansley, S., & Bethell, O. (2023). Hallucination is the last thing you need.
- Dahl, M., Magesh, V., Suzgun, M., & Ho, D. E. (2024). Large legal fictions: Profiling legal hallucinations in large language models.
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. <https://arxiv.org/abs/1810.04805>
- Ding, H., Pang, L., Wei, Z., Shen, H., & Cheng, X. (2024). Retrieve only when it needs: Adaptive retrieval augmentation for hallucination mitigation in large language models.

- Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., Yang, F., & Yang, M. (2024). Longrope: Extending llm context window beyond 2 million tokens. <https://arxiv.org/abs/2402.13753>
- Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., Sun, X., Li, L., & Sui, Z. (2024). A survey on in-context learning. <https://arxiv.org/abs/2301.00234>
- Duque, A., Syed, A., Day, K. V., Berry, M. J., Katz, D. S., & Kindratenko, V. V. (2023). Leveraging large language models to build and execute computational workflows. <https://arxiv.org/abs/2312.07711>
- Faiz, A., Kaneda, S., Wang, R., Osi, R., Sharma, P., Chen, F., & Jiang, L. (2024). Llmcarbon: Modeling the end-to-end carbon footprint of large language models. <https://arxiv.org/abs/2309.14393>
- Gekhman, Z., Yona, G., Aharoni, R., Eyal, M., Feder, A., Reichart, R., & Herzig, J. (2024). Does fine-tuning llms on new knowledge encourage hallucinations?
- Gema, A. P., Minervini, P., Daines, L., Hope, T., & Alex, B. (2024). Parameter-efficient fine-tuning of llama for the clinical domain. <https://arxiv.org/abs/2307.03042>
- Gupta, S. (2020). Exploring neural net augmentation to bert for question answering on squad 2.0. <https://arxiv.org/abs/1908.01767>
- Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks. <https://arxiv.org/abs/2004.10964>
- Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M.-W. (2020). Realm: Retrieval-augmented language model pre-training.
- Han, Z., Gao, C., Liu, J., Zhang, J., & Zhang, S. Q. (2024). Parameter-efficient fine-tuning for large models: A comprehensive survey.
- He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., & Neubig, G. (2022). Towards a unified view of parameter-efficient transfer learning. <https://arxiv.org/abs/2110.04366>
- Houlsby, N., Giurui, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., & Gelly, S. (2019). Parameter-efficient transfer learning for nlp. <https://arxiv.org/abs/1902.00751>
- Hsieh, C.-Y., Chen, S.-A., Li, C.-L., Fujii, Y., Ratner, A., Lee, C.-Y., Krishna, R., & Pfister, T. (2023). Tool documentation enables zero-shot tool-usage with large language models.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). Lora: Low-rank adaptation of large language models.

- Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions.
- Iliopoulos, F., Kontonis, V., Baykal, C., Menghani, G., Trinh, K., & Vee, E. (2022). Weighted distillation with unlabeled examples. <https://arxiv.org/abs/2210.06711>
- Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., & Grave, E. (2022). Atlas: Few-shot learning with retrieval augmented language models.
- Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12), 1–38. <https://doi.org/10.1145/3571730>
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). Mistral 7b.
- Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. <https://arxiv.org/abs/1910.13461>
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.
- Li, C., Liang, J., Zeng, A., Chen, X., Hausman, K., Sadigh, D., Levine, S., Fei-Fei, L., Xia, F., & Ichter, B. (2023). Chain of code: Reasoning with a language model-augmented code emulator.
- Lialin, V., Shivagunde, N., Muckatira, S., & Rumshisky, A. (2023). Relora: High-rank training through low-rank updates.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). Lost in the middle: How language models use long contexts.
- Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., & Chen, M.-H. (2024). Dora: Weight-decomposed low-rank adaptation. <https://arxiv.org/abs/2402.09353>
- Min, S., Gururangan, S., Wallace, E., Hajishirzi, H., Smith, N. A., & Zettlemoyer, L. (2023). Silo language models: Isolating legal risk in a nonparametric datastore.

- Mosbach, M., Pimentel, T., Ravfogel, S., Klakow, D., & Elazar, Y. (2023). Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. <https://arxiv.org/abs/2305.16938>
- Munkhdalai, T., Faruqui, M., & Gopal, S. (2024). Leave no context behind: Efficient infinite context transformers with infini-attention. <https://arxiv.org/abs/2404.07143>
- OpenAI. (2024). Gpt-4 technical report.
- Ostapenko, O., Su, Z., Ponti, E. M., Charlin, L., Roux, N. L., Pereira, M., Caccia, L., & Sordoni, A. (2024). Towards modular llms by building and reusing a library of lorae.
- Pandey, R., Purohit, H., Castillo, C., & Shalin, V. L. (2022). Modeling and mitigating human annotation errors to design efficient stream processing systems with human-in-the-loop machine learning. *International Journal of Human-Computer Studies*, 160, 102772. <https://doi.org/10.1016/j.ijhcs.2022.102772>
- Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network training.
- Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., & Sun, M. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis.
- Sarkar, A., Gordon, A. D., Negreanu, C., Poelitz, C., Ragavan, S. S., & Zorn, B. (2022). What is it like to program with artificial intelligence? <https://arxiv.org/abs/2208.06213>
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. <https://arxiv.org/abs/2302.04761>
- Semnani, S. J. (2019). Bert-a : Fine-tuning bert with adapters and data augmentation. <https://api.semanticscholar.org/CorpusID:204735588>
- Shen, W., Li, C., Chen, H., Yan, M., Quan, X., Chen, H., Zhang, J., & Huang, F. (2024). Small llms are weak tool learners: A multi-llm agent.
- Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E., Schärli, N., & Zhou, D. (2023). Large language models can be easily distracted by irrelevant context. <https://arxiv.org/abs/2302.00093>

- Shi, Z., Gao, S., Chen, X., Feng, Y., Yan, L., Shi, H., Yin, D., Chen, Z., Verberne, S., & Ren, Z. (2024). Chain of tools: Large language model is an automatic multi-tool learner.
- Sun, C., Huang, L., & Qiu, X. (2019). Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence. <https://arxiv.org/abs/1903.09588>
- Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., & Sun, L. (2023). Toolalpaca: Generalized tool learning for language models with 3000 simulated cases.
- Team, G., Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G., Vincent, D., Pan, Z., Wang, S., Mariooryad, S., Ding, Y., Geng, X., Alcober, F., Frostig, R., Omernick, M., Walker, L., Paduraru, C., Sorokin, C., . . . Vinyals, O. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. <https://arxiv.org/abs/2403.05530>
- Team, L. (2024). The llama 3 herd of models. *arXiv*. <https://ai.meta.com/research/publications/the-llama-3-herd-of-models/>
- Torrey, L., & Shavlik, J. (2009). *Transfer learning*. <https://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf>
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., . . . Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention is all you need.
- Vemprala, S., Bonatti, R., Bucker, A., & Kapoor, A. (2023). Chatgpt for robotics: Design principles and model abilities. <https://arxiv.org/abs/2306.17582>
- Wang, Y., Agarwal, S., Mukherjee, S., Liu, X., Gao, J., Awadallah, A. H., & Gao, J. (2022, December). AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In Y. Goldberg, Z. Kozareva, & Y. Zhang (Eds.), *Proceedings of the 2022 conference on empirical methods in natural language processing* (pp. 5744–5760). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2022.emnlp-main.388>
- Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., Pathak, E., Karamanolakis, G., Lai, H. G., Purohit, I., Mondal, I., Anderson, J., Kuznia, K., Doshi, K., Patel, M., . . . Khashabi, D. (2022). Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. <https://arxiv.org/abs/2204.07705>

- Wang, Y., Sun, Y., Ma, Z., Gao, L., Xu, Y., & Sun, T. (2020). Application of pre-training models in named entity recognition. <https://arxiv.org/abs/2002.08902>
- Wang, Z., Cheng, Z., Zhu, H., Fried, D., & Neubig, G. (2024). What are tools anyway? a survey from the language model perspective.
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.
- Xiong, W., Liu, J., Molybog, I., Zhang, H., Bhargava, P., Hou, R., Martin, L., Rungta, R., Sankararaman, K. A., Oguz, B., Khabsa, M., Fang, H., Mehdad, Y., Narang, S., Malik, K., Fan, A., Bhosale, S., Edunov, S., Lewis, M., ... Ma, H. (2023). Effective long-context scaling of foundation models.
- Xu, L., Xie, H., Qin, S.-Z. J., Tao, X., & Wang, F. L. (2023). Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment.
- Xu, Q., Hong, F., Li, B., Hu, C., Chen, Z., & Zhang, J. (2023). On the tool manipulation capability of open-source large language models. <https://arxiv.org/abs/2305.16504>
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). React: Synergizing reasoning and acting in language models.
- Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., Zhang, K., Ji, C., Yan, Q., He, L., Peng, H., Li, J., Wu, J., Liu, Z., Xie, P., Xiong, C., Pei, J., Yu, P. S., & Sun, L. (2023). A comprehensive survey on pretrained foundation models: A history from bert to chatgpt.
- Zhuang, Y., Yu, Y., Wang, K., Sun, H., & Zhang, C. (2023). Toolqa: A dataset for llm question answering with external tools.

Appendix A

Model Glossary

On the next page

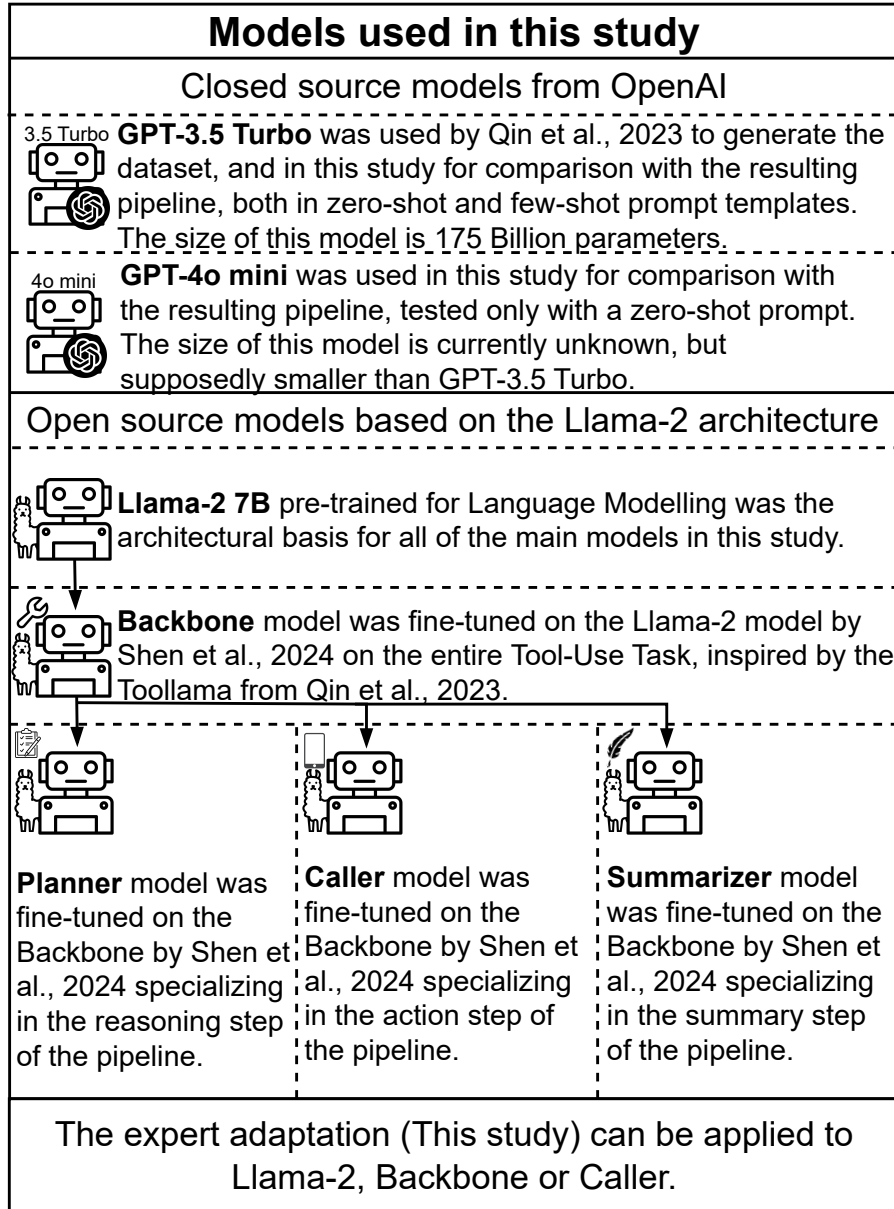


Figure A.1: Models used in this study with description of their origin and purpose.

Appendix B

Chart Data

B.1 Methodology chapter

B.1.1 Training Dataset

On the next page

Category Name	Number of	
	samples	tasks
data	38,245	1,460
finance	30,100	1,451
entertainment	27,364	564
social	20,397	724
tools	20,868	534
sports	21,704	1,207
news_media	11,857	317
travel	11,019	373
business	10,062	409
commerce	5,847	118
ecommerce	9,642	302
music	9,574	412
weather	9,452	214
education	9,440	206
location	8,868	346
other	8,477	447
food	8,338	194
gaming	7,316	309
database	7,109	227
mapping	6,670	120
communication	6,582	218
search	6,080	124
financial	5,488	186
video_images	5,187	202
science	4,849	98
media	4,752	121
translation	4,212	75
business_software	4,175	187
advertising	4,096	119
text_analysis	3,647	91
movies	3,533	113
health_and_fitness	2,832	85
AI_ML	2,776	89
transportation	2,424	137
sms	2,347	69
medical	2,258	45
email	2,243	118
events	2,094	52
monitoring	1,654	42
logistics	1,386	33
visual_recognition	1,261	38
category_not_found	1,229	62
devices	961	32
energy	563	20
cybersecurity	429	12
jobs	407	12
reward	337	10
payments	304	27
storage	281	6
cryptography	242	17
customized	58	2

Table B.1: Distribution of categories in the ToolBench dataset, including the number of samples and the count of endpoints. AI_ML is a shorthand for the artificial_intelligence_machine_learning category.

B.2 Results chapter

B.2.1 Optimal Hierarchical Level of Abstraction

Short Name	Full Name
JCN_R	jokes_random_for_chuck_norris
JCN_S	jokes_search_for_chuck_norris
JCN_C	jokes_categories_for_chuck_norris
DZ_S	search_for_deezer
DZ_P	playlist_for_deezer
DZ_G	genre_for_deezer
DZ_I	infos_for_deezer
DZ_E	editorial_for_deezer
DZ_A	artist_for_deezer
DZ_T	track_for_deezer
DZ_R	radio_for_deezer
DZ_AL	album_for_deezer
DZ_C	comment_for_deezer
N_RF	get_random_fact_for_numbers
N_TF	get_trivia_fact_for_numbers
N_MF	get_math_fact_for_numbers
N_DF	get_date_fact_for_numbers
N_YF	get_year_fact_for_numbers

Table B.2: Disambiguation table for short names used in the "Optimal Hierarchical Level of Abstraction" experiment.

API	API Ch. (EM)				Arg. Pop. (F1)			
	Cal	Cat	API	End	Cal	Cat	API	End
JCN_R	0.35	0.34	0.92	0.90	0.35	0.34	0.91	0.90
JCN_S	0.33	0.33	0.90	0.90	0.31	0.32	0.86	0.85
JCN_C	0.46	0.46	0.91	0.93	0.46	0.46	0.91	0.93
DZ_S	0.27	0.26	0.89	0.58	0.18	0.18	0.79	0.47
DZ_P	0.30	0.34	0.86	0.84	0.12	0.16	0.68	0.64
DZ_G	0.43	0.30	0.87	0.87	0.22	0.16	0.77	0.77
DZ_I	0.29	0.29	0.88	0.85	0.29	0.29	0.88	0.85
DZ_E	0.40	0.36	0.91	0.96	0.23	0.19	0.68	0.76
DZ_A	0.21	0.17	0.81	0.83	0.10	0.07	0.62	0.63
DZ_T	0.33	0.25	1.00	1.00	0.21	0.15	0.83	0.81
DZ_R	0.48	0.84	0.84	0.88	0.18	0.58	0.58	0.62
DZ_AL	0.16	0.84	0.84	0.84	0.08	0.62	0.62	0.62
DZ_C	0.00	0.12	0.75	0.75	0.00	0.06	0.62	0.62
N_RF	0.26	0.26	0.88	0.86	0.23	0.23	0.83	0.82
N_TF	0.29	0.30	0.84	0.82	0.18	0.23	0.80	0.78
N_MF	0.24	0.90	0.88	0.90	0.15	0.88	0.85	0.88
N_DF	0.15	0.07	0.93	0.93	0.11	0.06	0.89	0.89
N_YF	0.21	0.84	0.87	0.87	0.09	0.84	0.87	0.86

Abbreviation	Meaning
API	API Endpoint (Tool)
API Ch. (EM)	API Choice (Exact Match)
Arg. Pop. (F1)	Argument Population (F1)
Cal	Caller
Cat	Category Expert
API	API Expert
End	Endpoint Experts

Table B.3: Comparison of hierarchical expert splits in terms of API choice (exact match) and argument population F1. Acronyms are disambiguated in Table B.2, and the information is illustrated in Graph 4.1.

B.2.2 Effect of Sample Number and the Tasks Considered

Task Name	Number of		Caller		Expert	
	Samples	Tasks	EM	F1	EM	F1
Data	38245	1460	0.37	0.30	0.36	0.30
Finance	30100	1451	0.28	0.23	0.31	0.25
Sports	21704	1270	0.39	0.34	0.39	0.35
Social	20397	724	0.31	0.26	0.34	0.28
Entertainment	27364	565	0.33	0.28	0.33	0.28
Tools	20868	534	0.58	0.53	0.37	0.32
Music	9574	412	0.50	0.43	0.88	0.80
Travel	11019	373	0.28	0.22	0.44	0.38
Education	9440	206	0.32	0.28	0.56	0.52
database	7109	227	0.54	0.55	0.49	0.50
billboard_api	2447	92	0.24	0.20	0.88	0.83
Jobs	407	12	0.30	0.28	0.59	0.57
numbers	2740	5	0.24	0.17	0.87	0.84
random_for_manatee_jokes	1070	1	0.37	0.37	0.93	0.93
search_for_soundcloud	1510	1	0.43	0.31	0.97	0.90
search_for_deezer	1239	1	0.27	0.18	0.58	0.47
jokes_random_for_chuck_norris	2362	1	0.31	0.29	0.90	0.85
exchange_for_currency_exchange	1006	1	0.43	0.32	0.42	0.28

Table B.4: Performance metrics from the comparison of the number of samples and tasks experiment. EM denotes the Exact Match on API choice, and the F1-score is measured based on the Argument Population. Data is illustrated in Figure 4.2.

B.2.3 Effect of Limiting the Dataset to Certain Plans

Short Name	Full Name
D	Data
CN	chuck_norris
BA	billboard_api
JCN	jokes_random_for_chuck_norris
TR	Transportation
SS	search_for_soundcloud
SD	search_for_deezer
RMJ	random_for_manatee_jokes
ECE	exchange_for_currency_exchange
GYF	get_year_fact_for_numbers

Table B.5: Disambiguation table for short names used in the "Effect of Limiting the Dataset to Certain Plans" experiment.

API	API Choice (Exact Match)		Argument Population (F1)	
	ALL	CERTAIN	ALL	CERTAIN
D	0.36	0.36	0.36	0.36
CN	0.91	0.38	0.90	0.38
BA	0.88	0.24	0.83	0.19
JCN	0.90	0.37	0.90	0.37
TR	0.65	0.33	0.61	0.27
SS	0.97	0.93	0.90	0.87
SD	0.58	0.30	0.47	0.20
RMJ	0.93	0.91	0.93	0.91
ECE	0.42	0.83	0.28	0.82
GYF	0.87	0.16	0.86	0.04

Table B.6: Effect of reducing "certain" entries on API choice and argument population (F1). Acronyms are disambiguated in Table B.5, and the information is shown in Graph 4.3.

B.2.4 Preferred Base Model Selection

Name	Base Model			Expert+Model		
	L	B	C	L	B	C
jokes_random_for_chuck_norris	0.00	0.43	0.35	0.00	0.43	0.90
search_for_soundcloud	0.01	0.06	0.31	0.00	0.07	0.90
random_for_manatee_jokes	0.00	0.24	0.37	0.00	0.24	0.93
search_for_deezer	0.00	0.10	0.43	0.02	0.06	0.90
exchange_for_currency_exchange	0.00	0.14	0.32	0.00	0.14	0.28
get_year_fact_for_numbers	0.00	0.07	0.08	0.00	0.09	0.86
chuck_norris	0.00	0.25	0.35	0.00	0.25	0.89
billboard_api	0.00	0.43	0.20	0.00	0.43	0.83
Data	0.00	0.17	0.30	0.00	0.14	0.30
Transportation	0.00	0.17	0.27	0.00	0.14	0.61

Table B.7: Argument Population F1 scores for tasks based on different base and expert models. "Base Model" denotes the model used without an adapter, while "Expert+Model" indicates the performance of the expert adapter when trained on LLAMA (L), Backbone (B), and Caller (C) models. This data is plotted in Figure 4.4.

B.2.5 Viability of Expert Aggregation for API Experts

Endpoint	Short Name	Aggregate		Direct		Endpoints	
		EM	F1	EM	F1	EM	F1
jokes_random_for_chuck_norris	JCN_R	0.87	0.87	0.92	0.91	0.90	0.90
jokes_search_for_chuck_norris	JCN_S	0.92	0.89	0.90	0.86	0.90	0.85
jokes_categories_for_chuck_norris	JCN_C	0.94	0.94	0.91	0.91	0.93	0.93
get_random_fact_for_numbers	N_RF	0.86	0.81	0.88	0.83	0.86	0.82
get_trivia_fact_for_numbers	N_TF	0.84	0.81	0.84	0.80	0.82	0.78
get_math_fact_for_numbers	N_MF	0.86	0.83	0.88	0.85	0.90	0.88
get_date_fact_for_numbers	N_DF	0.95	0.90	0.93	0.89	0.93	0.89
get_year_fact_for_numbers	N_YF	0.87	0.86	0.87	0.87	0.87	0.86
search_for_deezer	DZ_S	0.84	0.79	0.89	0.78	0.58	0.47
playlist_for_deezer	DZ_P	0.86	0.68	0.86	0.68	0.84	0.64
genre_for_deezer	DZ_G	0.87	0.77	0.87	0.77	0.87	0.77
infos_for_deezer	DZ_I	0.88	0.88	0.88	0.88	0.85	0.85
editorial_for_deezer	DZ_E	0.91	0.68	0.91	0.68	0.96	0.76
artist_for_deezer	DZ_A	0.81	0.62	0.81	0.62	0.83	0.63
track_for_deezer	DZ_T	1.00	0.83	1.00	0.83	1.00	0.81
radio_for_deezer	DZ_R	0.84	0.58	0.84	0.58	0.88	0.62
album_for_deezer	DZ_AL	0.84	0.62	0.84	0.62	0.84	0.62
comment_for_deezer	DZ_C	0.75	0.62	0.75	0.62	0.75	0.62

Table B.8: Performance metrics for various API endpoints, highlighting the impact of different methods. Abbreviations used are Aggregate (Aggregated API Expert), Direct (Directly Fine-tuned API Expert), and Endpoints (Endpoint Experts). Metrics are provided for API Choice Exact Match (EM) and F1-scores for Argument Population. This information is illustrated in Graph 4.5.

B.2.6 Progressive Adapter Stacking for Boosting Performance

Name	Acronym	DT	PST	SST
API Choice - Exact Match				
chuck_norris	CN	0.91	0.89	0.90
free_nba	FN	0.31	0.88	0.88
currency_exchange	CE	0.83	0.85	0.85
jokes_random_for_chuck_norris	JCN_R	0.90	0.86	0.88
jokes_search_for_chuck_norris	JCN_S	0.90	0.92	0.92
jokes_categories_for_chuck_norris	JCN_C	0.93	0.94	0.94
Argument Population - F1				
chuck_norris	CN	0.90	0.88	0.89
free_nba	FN	0.27	0.84	0.84
currency_exchange	CE	0.83	0.84	0.84
jokes_random_for_chuck_norris	JCN_R	0.90	0.86	0.88
jokes_search_for_chuck_norris	JCN_S	0.85	0.89	0.89
jokes_categories_for_chuck_norris	JCN_C	0.93	0.94	0.94

Table B.9: Scores for various tasks based on different training methods: DT (Direct Training), PST (Progressive Stacked Training), and SST (Simple Stacked Training). Metrics are provided for "API Choice - Exact Match" and "Argument Population - F1". Full names for acronyms are included in the new column. This information is plotted in Figure 4.6.

B.2.7 In-Domain and Out-Of-Domain Test and Comparison

Task	Caller (Shen et al., 2024)	GPT-4o mini (OpenAI)	GPT-3.5 Turbo (OpenAI) Zero-shot	GPT-3.5 Turbo (OpenAI) Few-shot	Caller+Expert Adapter (This study)
API Choice - Exact Match					
music	0.33	0.57	0.52	0.57	0.67
news_media	0.29	0.49	0.59	0.45	0.63
business	0.41	0.45	0.62	0.55	0.69
finance	0.35	0.64	0.62	0.64	0.38
entertainment	0.42	0.55	0.54	0.55	0.51
education	0.50	0.75	0.58	0.58	0.83
travel	0.38	0.42	0.54	0.54	0.62
tools	0.44	0.44	0.56	0.44	0.56
data	0.43	0.64	0.62	0.61	0.46
sports	0.33	0.53	0.60	0.53	0.65
transportation	0.33	0.73	0.73	0.73	0.80
social	0.44	0.75	0.67	0.69	0.83
Argument Population - F1					
music	0.24	0.27	0.31	0.43	0.47
news_media	0.23	0.26	0.55	0.42	0.57
business	0.39	0.24	0.56	0.55	0.65
finance	0.28	0.36	0.51	0.56	0.33
entertainment	0.39	0.27	0.47	0.51	0.47
education	0.50	0.25	0.54	0.58	0.75
travel	0.35	0.12	0.41	0.46	0.49
tools	0.41	0.20	0.49	0.42	0.52
data	0.35	0.31	0.51	0.54	0.37
sports	0.31	0.29	0.49	0.51	0.63
transportation	0.23	0.42	0.68	0.68	0.70
social	0.37	0.33	0.49	0.57	0.68

Table B.10: Scores for various tasks based on different models: API Choice - Exact Match and Argument Population - F1. The models include Caller (Shen et al., 2024), GPT-4o mini (OpenAI), GPT-3.5 Turbo (OpenAI) Zero-shot, GPT-3.5 Turbo (OpenAI) Few-shot, and Caller+Expert Adapter (this study). Data is displayed in Figure 4.7.

Task	Caller (Shen et al., 2024)	Caller+Expert Adapter (This study)
API Choice - Exact Match		
music	0.22	0.67
news_media	0.46	0.46
business	0.40	0.60
finance	0.75	0.75
entertainment	0.15	0.15
education	0.38	0.44
tools	0.51	0.49
data	0.56	0.56
sports	0.00	0.00
social	0.50	0.50
Argument Population - F1		
music	0.11	0.39
news_media	0.46	0.46
business	0.10	0.20
finance	0.38	0.56
entertainment	0.15	0.15
education	0.34	0.41
tools	0.47	0.47
data	0.44	0.39
sports	0.00	0.00
social	0.38	0.38

Table B.11: Scores for various tasks based on different models: API Choice - Exact Match and Argument Population - F1. The models include Caller (Shen et al., 2024) and Caller+Expert Adapter (This study). This data is shown in Figure 4.8.

Appendix C

Prompts Used

C.1 ToolAlpaca Test Set Categorization

Prompt Template used for categorizing ToolAlpaca (Tang et al., 2023) tools onto the mappings used in ToolBench (Qin et al., 2023).

```
Classify the API below to one of the categories (your
output is only the category name and it must be in the
set of categories): {API_FAMILY}: {(ENDPOINT => NAME)
[] } | Descriptions: {(ENDPOINT => DESCRIPTION) [] }.
Categories: {CATEGORY_NAMES [] }
```

An example of the Template shown in Appendix C.1.

Classify the API below to one of the categories (your output is only the category name and it must be in the set of categories):

```
geolocation_simulation_for_mobile_apps:
simulations_of_a_group, list_all_groups,
list_all_layers, list_all_simulations,
list_applications, ons_by_application_id,
view_application, view_layer, view_simulation,
view_user_s_profile | Descriptions: Enter application
Enter groupID Enter layerID Enter simulation Enter
your application List all your a View Profile (E
list_all_groups)
```

Categories: Logistics, Cryptography, Jobs, Gaming, Social
, Sports, Database, Business_Software, Music, Business
, Location, Travel,
Artificial_Intelligence_Machine_Learning, Science,
Email, Events, Health_and_Fitness, Payments, Movies,
Text_Analysis, Transportation, Monitoring, Medical,
Financial, Weather, Video_Images, Devices, Customized,
SMS, Food, Entertainment, Advertising, Energy, Tools,
Search, Media, eCommerce, Visual_Recognition, Data,
Communication, Other, Finance, Cybersecurity,
News_Media, Translation, Mapping, Commerce, Storage,
Reward, Education, Unknown

C.2 ToolBench Multi-Agent prompt structure

Template used in the Multi-Agent pipeline proposed by Shen et al., 2024.

You have access to the following apis: `TOOLS_RETRIVED: (TOOL => DOCUMENTATION) []` The conversation history is: `UTTERANCES: (UTTERANCE => TEXT) []` You should answer the action of next step in the following format: The thought to solve the question, Action: the name of api that should be called in this step, should be exactly in `[TOOLS_RETRIVED: (TOOL => TOOL_NAME) []]`, Action Input: the api call request. Once you think the question is finished, output conclusion: the final answer of the question or give up if you think you cannot answer this question.

An example of the use of the template shown above. This conversation comes from the ToolBench test set and it was generated by GPT-3.5 Turbo, Please note that this example has the conclusion included, which would not happen in reality as the LLM does not need to inspect the conversation at that point, but it was included so that the entire conversation can be inspected.

You have access to the following apis:

search_for_ott_details: This is the subfunction for tool "ott_details", you can use this tool. The description of this function is: "This endpoint allows you to search for a m"

params_for_ott_details: This is the subfunction for tool "ott_details", you can use this tool. The description of this function is: "Get array of values that can be used as pa"

basic_info_for_ott_details: This is the subfunction for tool "ott_details", you can use this tool. The description of this function is: "Get info on a cast such as name, profession"

sources_for_watchmode: This is the subfunction for tool "watchmode", you can use this tool. The description of this function is: "Return a listing of all streaming sources th"

new_titles_for_watchmode: This is the subfunction for tool "watchmode", you can use this tool. The description of this function is: "Get a listing of the title IDs of new titles"

titles_with_changed_episodes_for_watchmode: This is the subfunction for tool "watchmode", you can use this tool. The description of this function is: "Get a listing of titles that have changes to"

The conversation history is:

user: I'm organising a movie-themed event for my company and I need some information about the cast member with the peopleid 'nm0000375'. Can you fetch the name,

profession, birth year, and best titles associated with this cast member? Also, provide the streaming sources available in the US and Canada for these titles.

assistant: Given the current state and the API call to retrieve basic information for "ott_details" with the argument "peopleid" set to "nm0000375", my thought is that the user is looking for specific details about a cast member for their movie-themed event. By fetching the name, profession, birth year, and best titles associated with this cast member, we can provide the user with valuable information for their event planning. Additionally, the user is interested in knowing the streaming sources available in the US and Canada for these titles, which implies they want to ensure accessibility for their audience. Using the "ott_details" function with the provided peopleid will give us the necessary information to fulfill the user's request. Next: Caller.

caller: Action: basic_info_for_ott_details

Action Input: {
 "peopleid": "nm0000375"
}

observation: "{ 'peopleid': 'nm0000375', 'name': 'Robert Downey Jr.', 'profession': ['actor', 'producer', 'soundtrack'], 'knownForTitles': ['tt0371746', 'tt1300854', 'tt4154796', 'tt0988045'], 'birthYear': '1965', 'deathYear': '\\\\\\\\\\\\N', 'bio': 'Robert Downey Jr. has evolved into one of the most respected actors in Hollywood. With an amazing list of credits to his name, he has managed to stay new and fresh even after over four decades in the business.Downey was

born April 4, 1965 in Manhattan, New York, the son of writer, director and filmographer Robert Downey Sr. and actress Elsie Downey (née Elsie Ann Ford). Robert's father is of half Lithuanian Jewish, one quarter Hungarian Jewish, and one quarter Irish, descent, while Robert's mother was of English, Scottish, German, and Swiss-German ancestry. Robert and his sister, Allyson Downey, were immersed in film and the performing arts from a very young age, leading Downey Jr. to study at the Stagedoor Manor Performing Arts Training Center..."

assistant: Based on the previous assistant message, the current API call is to search for streaming details using the "ott_details" function with the argument "title" set to "tt0371746". My thought is that the user is interested in finding the streaming sources and availability of a specific title, "tt0371746", which is one of the best titles associated with the cast member requested earlier. By retrieving these streaming details, we can provide the user with information about where this title can be accessed, helping them plan their movie-themed event accordingly. Next: Caller.

caller: Action: search_for_ott_details
Action Input: {
 "title": "tt0371746"
}

observation:

assistant: Based on the previous action and its result, it seems like there was an error while searching for the OTT details of the title "tt1300854". Since this

action failed, I should try a different title to gather the streaming sources available in the US and Canada. I will now search for the OTT details of the title "tt4154796" to fetch the streaming sources. Next : Caller.

caller: Action: search_for_ott_details

Action Input: {
 "title": "tt4154796"
}

observation:

user: This is not the first time you try this task, all previous trails failed.

Before you generate my thought for this state, I will first show you your previous actions for this state, and then you must generate actions that is different from all of them. Here are some previous actions candidates:

```
[  
  {  
    "name": "Finish",  
    "arguments": "{  
      \"return_type\": \"give_up_and_restart\"  
    }",  
    "function_output": "{ \"response\": \"chose to give up  
      and restart\" }",  
    "mento-carlo-action-value": -10000  
  }  
]
```

again, your former observation: {"error": "request invalid, data error. status_code=500", "response": ""}

Remember you are now in the intermediate state of a trail

, you will first analyse the now state and previous action candidates, then make actions that is different from all the previous.

assistant: Based on the current state and previous actions, it seems that searching for the OTT details of the titles "tt0371746" and "tt4154796" resulted in an error. Since all previous actions failed, I should try a different approach to gather the streaming sources available in the US and Canada for the titles associated with Robert Downey Jr.

Instead of directly searching for the OTT details of a specific title, I can use the "sources_for_watchmode" function to get a listing of all streaming sources supported by Watchmode. This will provide me with the streaming sources available for the titles associated with Robert Downey Jr. in the US and Canada. Next: Caller.

caller: Action: sources_for_watchmode

Action Input: {}

observation: "[{'id': 203, 'name': 'Netflix', 'type': 'sub', 'logo_100px': 'https://cdn.watchmode.com/provider_logos/netflix_100px.png', 'ios_appstore_url': 'http://itunes.apple.com/app/netflix/id363590051', 'android_playstore_url': 'https://play.google.com/store/apps/details?id=com.netflix.mediaclient&hl=en', 'android_scheme': 'nflx', 'ios_scheme': 'nflx', 'regions': ['US', 'CA', 'GB', 'AU', 'BR', 'AR', 'BE', 'BG', 'CH', 'CL', 'CO', 'CZ', 'DE', 'DK', 'EC', 'EE', 'ES', 'FI', 'FR', 'GR', 'HK', 'HR', 'HU', 'ID', 'IE', 'IL', 'IN', 'IS', 'JP', 'KR', 'LT', 'MX', 'MY', 'NL', 'NO', 'NZ', 'PA', 'PE', 'PH', 'PL', 'PT', 'RO', 'RS', 'RU', 'SE', 'SI', 'SK', 'SN', 'TH', 'TR', 'TW', 'UK', 'VN', 'ZA']}]"

```
SE', 'SG', 'TH', 'TR', 'UA', 'VN', 'ZA', 'RU']}, {'id': 157, 'name': 'Hulu', 'type': 'sub', 'logo_100px': 'https://cdn.watchmode.com/provider_logos/hulu_100px.png', 'ios_appstore_url': 'http://itunes.apple.com/app/hulu-plus/id376510438', 'android_playstore_url': 'https://play.google.com/store/apps/details?id=com.hulu.plus', 'android_scheme': 'hulu', 'ios_scheme': 'hulu', 'regio...'}
```

assistant: The streaming sources available in the US and Canada are Netflix, Hulu, and Disney+. These sources provide access to a wide range of titles. I was unable to fetch the specific titles associated with Robert Downey Jr., but you can search for his best titles on these streaming platforms to find the movies he is known for.

assistant: The user's previous action of searching for the OTT details of specific titles resulted in errors. To find the streaming sources available in the US and Canada for the titles associated with Robert Downey Jr., I should take a different approach. Instead of relying on a specific title, I can use the "sources_for_watchmode" function to gather a comprehensive listing of streaming sources supported by Watchmode. This will provide me with the desired information about the streaming availability of Robert Downey Jr.'s best titles. Next: conclusion.

conclusion: The cast member with the peopleid 'nm0000375' is Robert Downey Jr. He is an actor, producer, and soundtrack artist. He was born in 1965. Some of his best titles include Iron Man, Avengers: Endgame, and Sherlock Holmes. These titles are available for streaming on Netflix, Hulu, and Disney+ in the US and

Canada.

You should answer the action of next step in the following format:

The thought to solve the question,

Action: the name of api that should be called in this step, should be exactly in [search_for_ott_details, params_for_ott_details, basic_info_for_ott_details, sources_for_watchmode, new_titles_for_watchmode, titles_with_changed_episodes_for_watchmode],

Action Input: the api call request.

Once you think the question is finished, output conclusion: the final answer of the question or give up if you think you cannot answer this question.