Function Calling in Large Language Models: Industrial Practices, Challenges, and Future Directions

MAOLIN WANG[*], City University of Hong Kong, China
YINGYI ZHANG[*], City University of Hong Kong, China
CUNYIN PENG[*], Ant Group, China
YICHENG CHEN, Ant Group, China
WEI ZHOU, Ant Group, China
JINJIE GU, Ant Group, China
CHENYI ZHUANG[†], Ant Group, China
RUOCHENG GUO, Unaffliated, China
BOWEN YU, City University of Hong Kong, China
WANYU WANG, City University of Hong Kong, China
XIANGYU ZHAO[†], City university of Hong Kong, China

The swift evolution of Large Language Models (LLMs) like the GPT family, LLaMA, ChatGLM, and Qwen represents significant progress in artificial intelligence research. Despite their remarkable capabilities in generating content, these models encounter substantial challenges when producing structured outputs and engaging in dynamic interactions, particularly when they need to retrieve external information in real time. To address these limitations, researchers have developed the "Function Calling" paradigm. This approach enables language models to analyze user inquiries and engage with defined functions, thereby facilitating precise responses through connections to external sources including databases, programming interfaces, and live data streams. This functionality has been successfully implemented across numerous sectors such as finance analytics, healthcare systems, and service operations.The implementation of function calling comprises three essential phases: preparation, execution, and processing. The preparation phase encompasses query analysis and function identification. During execution, the system evaluates whether a function is necessary, extracts relevant parameters, and oversees the operation. The processing phase concentrates on analyzing outcomes and crafting appropriate responses. Each phase presents unique difficulties, ranging from accurately selecting functions to managing complex parameter extraction and ensuring reliable execution. Researchers have established various evaluation frameworks and metrics to assess function calling performance, including success rates, computational efficiency, parameter extraction accuracy, and response

[*]Equal Contribution
[†]Corresponding Authors

Authors' Contact Information: Maolin Wang, morin.wang@my.cityu.edu.hk, City University of Hong Kong, Hong Kong, China; Yingyi Zhang, yzhang6375-c@my.cityu.edu.hk, City University of Hong Kong, Hong Kong, China; Cunyin Peng, pengcunyin@gmail.com, Ant Group, Hangzhou, China; Yicheng Chen, chenggejiayou@gmail.com, Ant Group, Hangzhou, China; Wei Zhou, fayi.zw@antgroup.com, Ant Group, Hangzhou, China; Jinjie Gu, jinjie.gujj@antgroup.com, Ant Group, Hangzhou, China; Chenyi Zhuang, chenyi.zcy@antgroup.com, Ant Group, Hangzhou, China; Ruocheng Guo, rguo.asu@gmail.com, Unaffliated, Hong Kong, China; Bowen Yu, bowyu2-c@my.cityu.edu.hk, City University of Hong Kong, Hong Kong, China; Wanyu Wang, wanyuwang4-c@my.cityu.edu.hk, City University of Hong Kong, Hong Kong, China; Xiangyu Zhao, xy.zhao@cityu.edu.hk, City university of Hong Kong, Hong Kong, China.

quality indicators such as ROUGE-L evaluation scores. This survey systematically reviews the current landscape of function calling in LLMs, analyzing technical challenges, examining existing solutions, and discussing evaluation methodologies. We particularly focus on practical implementations and industrial applications, providing insights into both current achievements and future directions in this rapidly evolving field. For more resources and a comprehensive collection of related research papers, please refer to our repository at GitHub [1].

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Natural language processing**; • **Information systems** → **Specialized information retrieval**.

Additional Key Words and Phrases: Large Language Models, Function Calling, Industrial Perspective, LLM Agent

ACM Reference Format:
Maolin Wang, Yingyi Zhang, Cunyin Peng, Yicheng Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang, Ruocheng Guo, Bowen Yu, Wanyu Wang, and Xiangyu Zhao. 2025. Function Calling in Large Language Models: Industrial Practices, Challenges, and Future Directions. 1, 1 (April 2025), 41 pages.

1 INTRODUCTION

Recent advancements in artificial intelligence have ushered in a transformative era with the development of large language models (LLMs) such as GPT series, LLama [164], ChatGLM [33, 212] and Qwen [10]. These models, particularly effective in generation tasks due to their robust generative capabilities, have established themselves as vital for complex language tasks where traditional methods struggle. However, despite these advancements, LLMs face significant challenges in specific applications, particularly in generating highly structured outputs and engaging in timely real-world interactions [123, 137]. Even with evolving technology, LLMs often generate incorrect or outdated results for scenarios outside their training scope [32, 57], indicating certain limitations. For example, asking a LLM to analyze the latest stock market prices would be ineffective as it cannot access or retrieve real-time market data. This illustrates the model's limitation in handling dynamic information that changes frequently.

To overcome these deficiencies and more effectively leverage the potential of LLMs in enterprise and technical domains, the concept of "Function Calling" has been introduced [61]. In this context, a **Function** is defined as a predefined block of code, software, or tools that accept parameters and return a result. **The term Function Calling in the context of Large Language Models describes their proficiency at interpreting user requests and employing designated computational procedures to deliver precise, contextually suitable answers**. This technological advancement substantially improves these AI systems' accuracy when processing sophisticated inquiries while facilitating direct connections with numerous external resources—including data repositories, streaming information feeds, external application interfaces, alternative sophisticated language models, and additional services. To illustrate, these systems can acquire current weather conditions through meteorological service integrations and implement programming fragments using flexible code interpretation mechanisms. Additionally, this capability supports instantaneous analytical processes and operational decisions crucial for various professional domains including financial research [106], medical information systems [53], and tailored support solutions [54, 60]. As a result, this innovation fundamentally transforms the architecture of automated intelligence platforms and their engagement patterns with human operators. For example, Wolfram Alpha's [54] plugin for ChatGPT leverages function calling to enable complex computational queries and data retrieval. Users can ask detailed mathematical, scientific, or statistical questions, and LLMs call Wolfram Alpha's API to

[1] https://github.com/Applied-Machine-Learning-Lab/Awesome-Function-Callings

provide accurate and up-to-date answers directly within ChatGPT. This significantly enhances the ability of LLMs to solve specific mathematical problems.

Thus, a significant ongoing task is to enhance LLMs, which are primarily trained in natural language, to develop robust capabilities for specialized function calling. As shown in Figure 1, for a comprehensive LLM function calling process, the entire process can be divided into three stages based on the timing of the call: pre-call, on-call, and post-call. In the pre-call stage, the model must pre-process the user's query, including planning and decomposing tasks and selecting suitable function candidates from a function pool. Next, when entering the on-call stage, the model needs to assess whether a function call is necessary to complete the current user's task, involving the correct triggering of a single function or multi-functions. This is because not every user query requires a function call; sometimes, standard dialogue suffices. The challenge in this stage is avoiding unnecessary function calls and ensuring that needed function calls occur at the right time. Once a function call is triggered, the model first extracts parameters, identifying specific parameters from the user's query and incorporating them into the function. Common practices include matching and copying, but user language often contains a large amount of pronouns and unstructured expressions. For instance, asking about "today" involves a date that changes daily. To address this, we can use adapters or automatic rewriting in an intermediate language to optimize the process. Another major challenge in function calling is that if the required parameter volume is not provided, the model might need to ask the users to obtain the necessary information. Once all necessary parameters are prepared, the LLM will send the output results for the next step of processing. In the post-call stage, since the LLM output is in a natural language format and differs from real-world functions, the LLM output generated in natural language needs to be mapped to executable functions in the real physical world before being sent to the server for execution. During the execution stage, natural challenges include handling function execution failures and discrepancies between the function's results and the query.

Nonetheless, evaluating the effectiveness of function calling involves various strategies [205] and various aspects [204]. Metrics such as pass rate and win rate [120], average time per call [150], parameter identification ratio [205], retrieval metrics (such as Recall@K, NDCG@K, COMP@K) [122] and ROUGE-L score [77] provide a comprehensive assessment of the LLM's performance in executing function calls. Each of these metrics provides valuable insights into different aspects of the function calling process, but a holistic evaluation requires considering all these factors together to fully understand the performance and limitations of LLMs in real-world applications. Numerous studies, including API-BLEND [13], Seal-Tools [183], ToolBench2 [120], APIBank [70], ToolEyes [204], Rotbench [205], and others, have conducted extensive evaluations of the function calling capabilities of LLMs, encompassing a wide range of aspects.

Although many surveys [90, 123, 179] have addressed the topics of function calling, tool learning, and tool use, they often lack detailed discussions on practical challenges and underlying motivations. For example, Qu et al. [123] provide a comprehensive framework for tool learning, covering stages such as task planning, tool selection, invocation, and response generation. However, their work does not explore how these workflows can be optimized for real-world multi-task scenarios, a crucial aspect for industrial applications. Similarly, Mialon et al. [90] discuss foundational definitions and workflows, focusing on how tools enhance reasoning and problem-solving capabilities in LLMs. However, they overlook the challenges of implementing function calling systems in practice, such as managing latency, tool selection errors, or system failures. Wang (2024 B) et al. [179] categorize tool learning techniques and evaluation metrics but do not address strategies for handling dynamic multi-tool environments or deployment at scale.

Yang et al. [200] and Wang (2023) et al. [180] broaden the discussion by emphasizing the importance of LLM interactions in decision-making and evaluation methodologies, respectively. Yang et al. focus on combining planning with tool invocation in complex tasks but limit their exploration to methodological overviews. Similarly, Wang (2023)

Table 1. Evaluation of Survey Papers Across Different Dimensions

Survey Paper	Definition or Workflow	Industrial Challenges	Sample Construction	Deployment & Inference	Evaluation	Futures & Outlook
Zhao et al. [220]	✓	✗	✗	✗	✓	✓
Gao et al. [40]	✓	✗	✗	✗	✓	✗
Xi et al. [188]	✓	✗	✗	✗	✓	✓
Sun et al. [157]	✓	✗	✗	✗	✓	✗
Qiao et al. [117]	✓	✗	✗	✗	✓	✗
Huang et al. [52]	✓	✗	✗	✗	✓	✗
Zhao et al. [221]	✓	✗	✗	✗	✓	✗
Wang (2023) et al. [180]	✓	✗	✓	✗	✓	✓
Wang (2024 A) et al. [170]	✓	✗	✗	✗	✓	✓
Wang (2024 B) et al. [179]	✓	✗	✗	✗	✓	✓
Qin et al. [210]	✓	✓	✓	✗	✓	✓
Yang et al. [200]	✓	✓	✓	✓	✗	✓
Mialon et al. [90]	✓	✗	✗	✗	✓	✗
Qu et al. [123]	✓	✗	✓	✗	✓	✓
Ours	✓	✓	✓	✓	✓	✓

et al. [180] highlight human-feedback-driven tool invocation but fail to provide solutions for practical challenges like invocation failures or latency in real-world applications. These works represent an important intermediate step but leave critical gaps in practical implementation. Broader surveys, such as Zhao et al. [221] and Huang et al. [52], explore general LLM techniques and planning capabilities without directly addressing tool learning or function calling. Reasoning surveys like Qiao et al. [117] and Sun et al. [157] overlook the integration of tools and workflows for solving complex tasks. Additionally, works on autonomous agents by Wang (2024 B) et al. [170] and Xi et al. [188] touch upon tools' importance but lack systematic discussions of invocation challenges. Lastly, retrieval-augmented generation is reviewed comprehensively by Gao et al. [40] and Zhao et al. [220], but their focus remains on retrieval rather than broader tool learning frameworks. We also recognize the foundational contributions of earlier works like Qu et al. [123] and Mialon et al. [90], which laid the groundwork for tool learning by defining workflows and highlighting its potential. Yang et al. [200] and Wang (2023) et al. [180] expand the scope to interactive tasks, reinforcing function calling as a vital direction for LLM research.

Building on these studies, our work addresses the gaps in systematic implementation and industrial deployment of function calling and tool learning. We conduct experiments to explore how LLMs can efficiently select and invoke tools in multi-task settings, while analyzing the influence of task complexity on tool selection mechanisms. We further propose practical strategies, such as dynamic task queuing to optimize tool invocation efficiency, RL-based approaches to improve adaptability to invocation failures, and lightweight interfaces to mitigate latency issues. As shown in Table 1, while prior works have extensively covered function calling definitions and evaluation strategies, few studies have comprehensively addressed industrial challenges, sample construction, and system deployment aspects. Our work aims to fill these gaps by providing a complete treatment across all dimensions, including practical implementation considerations and deployment strategies.

In this paper, we make several key contributions to the field of LLMs function calling:

- We clearly define the concept of function calling and provide a comprehensive function calling pipeline.
- We describe in detail the potential challenges encountered at each step of the function calling pipeline.
- We outline strategies for sample construction and training as well as model deployment to tackle all identified challenges. Importantly, we validate several of these strategies through carefully designed experiments, demonstrating their effectiveness.

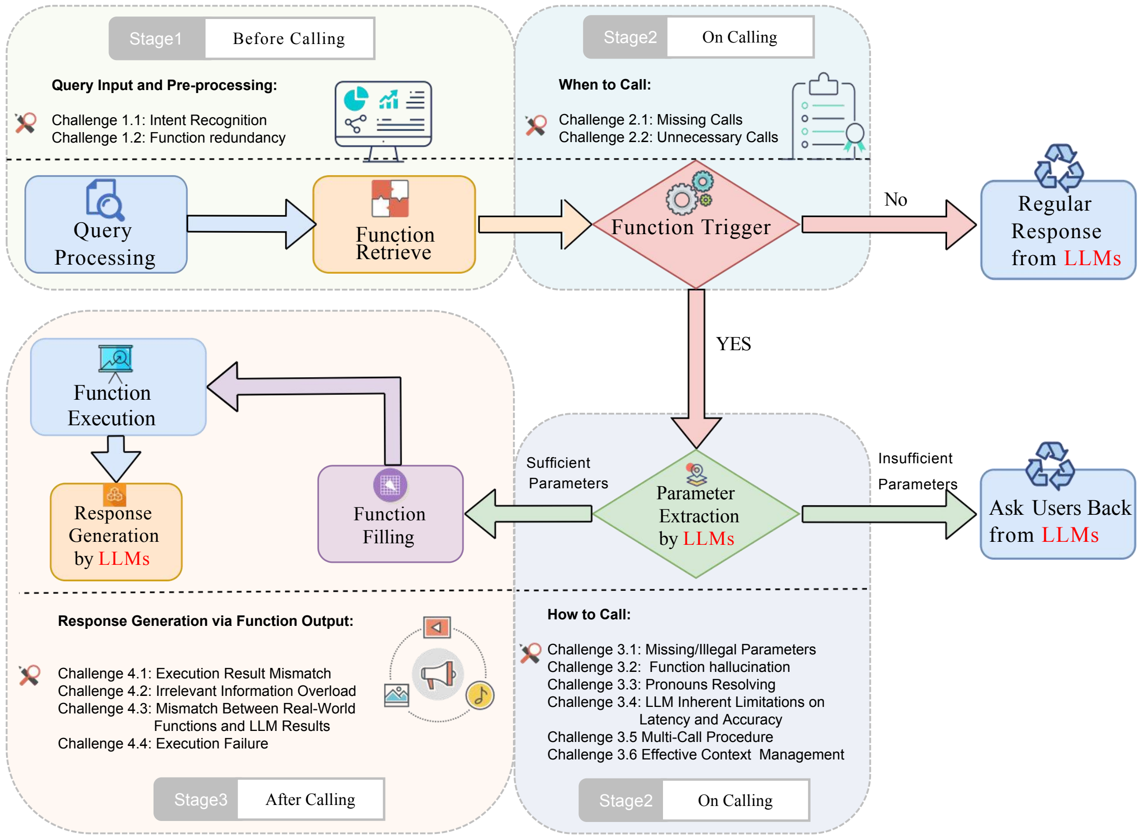
Stage1 Before Calling
Query Input and Pre-processing:
Challenge 1.1: Intent Recognition
Challenge 1.2: Function redundancy
Query Processing
Function Retrieve
Stage2 On Calling
When to Call:
Challenge 2.1: Missing Calls
Challenge 2.2: Unnecessary Calls
Function Trigger
No
Regular Response from LLMs
YES
Parameter Extraction by LLMs
Sufficient Parameters
Insufficient Parameters
Ask Users Back from LLMs
Function Filling
Function Execution
Response Generation by LLMs
Response Generation via Function Output:
Challenge 4.1: Execution Result Mismatch
Challenge 4.2: Irrelevant Information Overload
Challenge 4.3: Mismatch Between Real-World Functions and LLM Results
Challenge 4.4: Execution Failure
Stage3 After Calling
How to Call:
Challenge 3.1: Missing/Illegal Parameters
Challenge 3.2: Function hallucination
Challenge 3.3: Pronouns Resolving
Challenge 3.4: LLM Inherent Limitations on Latency and Accuracy
Challenge 3.5 Multi-Call Procedure
Challenge 3.6 Effective Context Management
Stage2 On Calling

Fig. 1. Typical function calling pipeline and associated challenges. The pipeline consists of three main stages: pre-call (including query processing and function retrieval), on-call (covering function triggering and parameter handling), and post-call (involving function execution and response generation). Each stage faces distinct challenges ranging from intent recognition and function redundancy in the pre-call stage to parameter extraction and multi-call procedures during execution to result in mismatch and execution failures in the post-call stage. LLMs are typically utilized in Stage 2 and Stage 3.

- We discuss comprehensive evaluation strategies for function calling and metrics.
- We also summarize the main datasets related to function calling in LLMs.
- We discuss future challenges and outlooks in this field.

In this paper, we present a comprehensive taxonomy of function calling in LLMs, which systematically categorizes key technical components and methodologies across the full lifecycle, from training data construction and model fine-tuning to deployment strategies and evaluation frameworks, providing a structured overview of both academic research advances and industrial applications in this rapidly evolving field. We have organized our survey paper into the following sections: Section 2 explores the motivations behind function calling and analyzes it through the function calling workflow, highlighting the associated challenges. In Section 3, we detail strategies from the perspectives of sample construction and model fine-tuning that are designed to overcome specific challenges in function calling. Subsequently, Section 4 delves into strategies for tailored system deployment and model inference to tackle particular challenges associated with function calling. Then, Section 5 discusses potential future developments and the corresponding challenges.

Additionally, we also provide supplementary information in the Appendix. **Appendix Section** A presents methods for assessing the performance of function calling. **Appendix Section** B examines existing industrial products along with essential datasets and benchmarks.

2 CONSTRUCTING THE FOUNDATION: FUNCTION CALLING PIPELINE AND CHALLENGES

We systematically analyze the function calling pipeline and its associated challenges in Section 2, providing a comprehensive framework for understanding this crucial aspect of LLM capabilities. This section presents a structured classification framework for identifying difficulties encountered throughout the function calling process. Our taxonomy systematically organizes these issues across multiple operational stages—beginning with query understanding prior to function invocation and extending through response management following execution completion.

2.1 Motivation

The incorporation of function calling mechanisms within Large Language Models serves to address inherent constraints these systems face beyond textual production capabilities [16]. Despite their sophisticated linguistic comprehension and generation faculties [168], these models exhibit notable limitations regarding real-time information acquisition, specialized computational operations, and external system interactions. Function calling integration enables these models to access designated interfaces and predefined procedural elements, thereby facilitating current data retrieval, complex operational execution, and fluid integration with complementary technological frameworks [105]. This technological advancement substantially expands the operational parameters of these systems while considerably improving both utilization experience and practical implementation value [21].

2.2 Pre-call Stage

Preliminary analysis of user inquiries and identification of applicable functional components must be systematically performed to guarantee optimal procedural selection while simultaneously maximizing response precision and operational efficiency throughout the interaction process.

2.2.1 ***Query Processing.*** The preliminary phase within the standard function calling operational sequence involves the model's systematic preprocessing of user-submitted inquiries. The core tasks of this stage include understanding the user's request and decomposing the task. Through this process, the model can identify key information and action steps that require further processing, preparing for the subsequent function retrieval stage. A significant challenge in this stage is:

Challenge 1.1: Intent Recognition The initial challenge lies in recognizing user intent to guide function calling.

Understanding user intent is crucial for addressing this challenge. Interpreting user requests to discern their intentions necessitates learning a mapping from the instruction space to the model's cognition space [155, 216]. These approaches employ external user-specific modules or prompts to incorporate users' preferences, styles, and personal information into the generated content. For example, GeckOpt [37] refines tool selection by incorporating intent-driven gating. Additionally, some research [185, 210] advocates for leveraging user feedback to dynamically adapt the model to individual users, presenting a promising avenue for personalizing LLMs. To the best of our knowledge, recognizing user intent in most function-calling applications predominantly relies on the inherent capabilities of large models and the strategic construction of prompts.

2.2.2 ***Function Retrieval Processing.*** After query processing, it is crucial to select relevant function candidates for subsequent steps. Given that a large commercial enterprise may have thousands of functions, failing to perform an initial screening could result in an overwhelming number of choices. This introduces a new challenge:

Challenge 1.2 Function redundancy: When multiple functions serve similar purposes, the system faces decreased efficiency and slower response times due to redundant function choices that complicate the selection process.

Initial function retrieval typically leverages the features of functions to address this challenge. It is important to distinguish between function selection and initial function retrieval: function selection involves choosing the correct function from a set of candidates, whereas retrieval pertains to identifying potential candidates. Some papers conflate these concepts. In this paper, 'initial function retrieval' denotes explicitly the process of identifying potential candidates. A practical retrieval module is crucial for selecting the top-K suitable function candidates from a large pool in scenarios with numerous functions. The introduction of this module bridges the gap between the capabilities of LLMs and practical input size limitations, as not all functions can be directly input into LLMs. Traditional term-based methods, such as BM25 [132], represent documents and queries as high-dimensional sparse vectors. For instance, Gorilla [113] combines BM25 and GPT-Index to construct a tool retriever for tool retrieval. Semantic similarity can be calculated using language model embeddings and cosine similarity. For instance, Confucius [39] trained a SentenceBERT model as a tool retriever, enhancing the efficiency of relevant tool retrieval. CRAFT [208] guides LLMs to generate fictitious tool descriptions based on a given query, utilizing these semantic details for searching. Similar to traditional information retrieval domains, some topological information can also be incorporated. For example, COLT [122] proposes a novel tool retrieval approach using Graph Neural Networks (GNNs) to ensure the completeness of the retrieved functions.

2.3 On-call Stage

After preparing the tasks and selecting a set of function candidates, the next typical step is to proceed with the function calling operation. This involves addressing two sub-problems: 'When to call' and 'How to call.' In practical implementations, the distinction between these two parts is often quite blurred, typically manifesting as a single-step generation by the LLM. However, to facilitate a comprehensive understanding of the entire function calling process, we discuss these two steps separately. Nevertheless, in practical applications, LLMs usually complete these steps **in a single operation** [105].

2.3.1 ***When to call.*** After function retrieval, the model needs to assess whether a function call is required to complete the user's task. This decision depends on the model's deep understanding of the task requirements and the context of the dialogue. Correct triggering is crucial to avoid unnecessary function calls and to ensure that necessary function calls occur timely. This presents two challenges:

Challenge 2.1: Missing Calls The LLM fails to initiate function calls when such actions are required. This lack of proper triggering can lead to missed opportunities for processing or accurately responding to user requests.

Challenge 2.2: Unnecessary Calls The LLM initiates unnecessary function calls when not required by the user's current task. As shown in ChemAgent's [207] evaluation, unnecessary function callings may underperform base LLMs on general tasks if used indiscriminately. This not only results in inefficiencies and a waste of resources but may also mislead the model's output.

2.3.2 ***How to call.*** Once a function call is triggered, the LLM needs to decide which function to call and extract the necessary parameters from the user's query. This process generally includes identifying keywords and phrases in the user's language and converting them into the parameters required for the correctly chosen function. This step is critical to ensuring the accuracy of the function call. However, the selection and extraction of parameters can encounter the following challenges:

Challenge 3.1: Missing/Illegal Parameters Missing/Illegal parameters issue arises when the parameters extracted from the user's input are inadequate or inappropriate for executing the intended function. This can lead to potential failures in function execution or the necessity for the model to solicit additional input from the user, thereby complicating the interaction and possibly affecting user satisfaction.

Challenge 3.2: Function hallucination The phenomenon of functional fabrication manifests when language models erroneously invoke non-applicable or entirely fictitious operational components, or populate parameters lacking legitimate existence. Such aberrations potentially generate inaccurate system responses and compromise procedural reliability, consequently inducing user dissatisfaction and diminishing confidence in the technological framework.

Furthermore, linguistic reference elements within user communications—including temporal indicators such as "today" and pronominal forms like "he" or "it"—necessitating contextual decoding for parameter determination, constitute an additional computational complexity.

Challenge 3.3 Pronouns Resolving: Accurate disambiguation of referential elements within user-initiated linguistic constructs represents a substantial technical impediment, attributable to the inherently fluid and context-contingent characteristics of natural language. Inadequate resolution of such referential ambiguities frequently precipitates erroneous parameter extraction mechanisms, thereby fundamentally compromising the operational integrity of function invocation protocols.

Moreover, consequent to their developmental conditioning within unconstrained textual corpora and intrinsically sophisticated architectural configurations, large language models manifest inherent computational constraints that significantly impact performance metrics, particularly regarding temporal response parameters and precision indicators. This operational limitation attains particular significance when these computational frameworks are deployed within specialized implementation domains necessitating exactitude and expeditious output generation.

Challenge 3.4: LLM Inherent Limitations on Latency and Accuracy: The fundamental operational limitations inherent within language modeling architectures—attributable to their developmental exposure to expansive yet non-domain-specific datasets coupled with their intricate algorithmic configurations—manifest as substantive impediments affecting response chronometry and interpretative fidelity when operationalized within specialized implementation contexts. Such algorithmic constraints frequently precipitate procedural latencies, interpretative aberrations, or critical analytical omissions throughout functional invocation sequences.

Furthermore, when confronted with user-initiated operational requisitions, the comprehensive fulfillment of such directives may necessitate sequential multiple-function activation protocols, exemplified by conference facility reservation processes wherein the computational framework must initially execute database interrogation procedures regarding facility availability parameters before subsequently implementing selection algorithms conforming to specified conditional criteria. This operational complexity introduces the following methodological challenge.

Challenge 3.5 Multi-Call Procedure: The procedural execution framework may mandate the preservation of comprehensive algorithmic comprehension across multiplicitous functional invocation mechanisms—whether manifested through concurrent parallel implementations or sequentially interdependent operational chains. Such computational orchestration necessitates sophisticated task decomposition methodologies while simultaneously administering complex dependency architectures throughout execution cycles.

Furthermore, consequent to the inherently iterative conversational structures characteristic of human-machine interactions and the potentially extensive epistemological repositories maintained by individual users, a significant methodological impediment emerges regarding informational continuity management.

Challenge 3.6 Effective Context Management: Comprehensive administration of this expansive informational continuity framework constitutes an operational imperative of paramount significance, functioning to mitigate potentialities of critical data attenuation while simultaneously ensuring the maintenance of response homogeneity throughout the duration of interactive procedural engagements. Inadequacies in this area can significantly impair the precision and user experience of function calling. Previous context may include essential user information, domain knowledge, or the history of multiple rounds of dialogue, including the results of functions executed in previous turns.

After parameter extraction, the model integrates these parameters into the chosen function. During this phase, the model may need to use built-in logic or interact with the user to address any parameter deficiencies. Once this step is completed, the model is prepared to send the function and its parameters, thus concluding the function calling process.

2.4 Post-call Stage

The objective of the post-call stage in the function-calling process is to execute the function and generate an appropriate natural language response. This phase involves converting the natural language outputs of the LLM into executable real-world functions. Once converted, these functions are sent for execution. Subsequently, the process also requires analyzing the execution results, managing instances of execution failures, and reconciling any discrepancies between the function's results and the user's original query and expectations. The challenges faced in this stage include:

Challenge 4.1 Execution Result Mismatch: Even when functions are correctly called, the outcomes may not align with user expectations, leading to mismatches in execution results. These mismatches can occur due to constraints inherent in the function's logic, malicious attacks [182, 213] on the function, or inconsistencies in the quality of function description.

Challenge 4.2 Irrelevant Information Overload: Some functions return some helpful, relevant information and an excessive amount of irrelevant information due to their design for comprehensiveness, which can lead to verbose responses and significant redundancy. This issue necessitates effective strategies for pruning irrelevant details to enhance the clarity and utility of the output.

Challenge 4.3 Mismatch Between Real-World Functions and LLM-Generated Results: Functions filled by an LLM generally cannot be directly executed due to a significant mismatch between the semantic space and the code space, making mapping a considerable challenge. For instance, to query the weather in "Hangzhou" on "January 1st, 2024", the function must be formatted correctly for the server. It may require a syntax like "GetWeather(date="2024.1.1", city="007")" instead of "GetWeather(date="January 1st, 2024", city="Hangzhou")" generated by LLMs.

Challenge 4.4 Execution Failure: Notwithstanding precise functional invocation and adequate parameterization protocols, operational execution failures may nevertheless manifest due to multifarious technical impediments. Execution anomalies potentially derive from server-side computational aberrations, structural incongruities between anticipated data architecture specifications and actual information configurational manifestations, or inherent constraints within functional design paradigms that preclude efficacious processing of boundary conditions or unanticipated input typologies. Such operational vulnerabilities necessitate the implementation of sophisticated exception management infrastructures and alternative procedural pathways to preserve systemic operational integrity and maintain user confidence metrics.

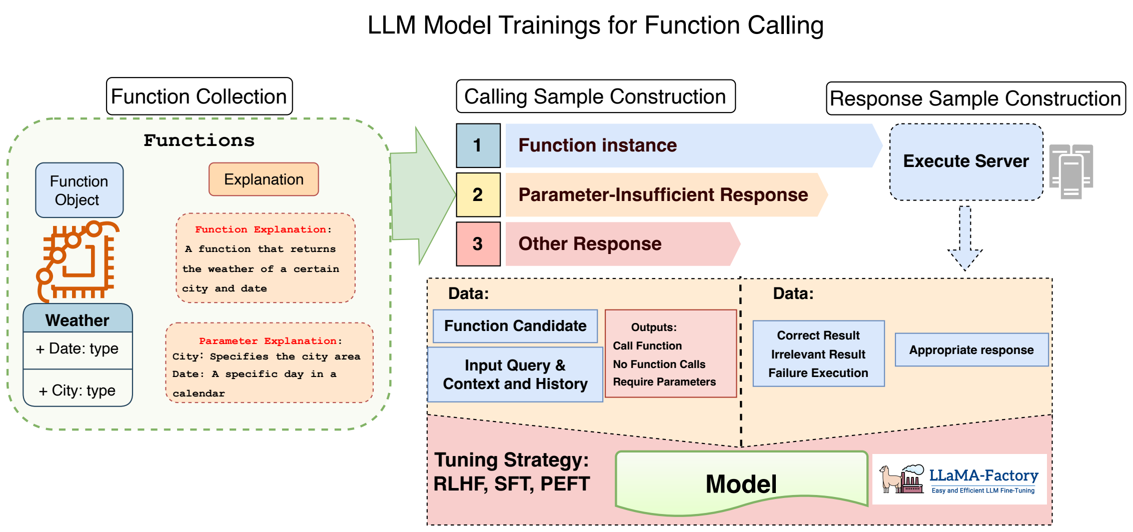
LLM Model Trainings for Function Calling
Function Collection
Functions
Function Object
Explanation
Function Explanation:
A function that returns the weather of a certain city and date
Weather
+ Date: type
+ City: type
Parameter Explanation:
City: Specifies the city area
Date: A specific day in a calendar
Calling Sample Construction
1 Function instance
2 Parameter-Insufficient Response
3 Other Response
Response Sample Construction
Execute Server
Data:
Function Candidate
Input Query & Context and History
Outputs:
Call Function
No Function Calls
Require Parameters
Data:
Correct Result
Irrelevant Result
Failure Execution
Appropriate response
Tuning Strategy: RLHF, SFT, PEFT
Model
LLaMA-Factory
Easy and Efficient LLM Fine-Tuning

Fig. 2. Exemplar Methodological Framework for Augmentation of Linguistic Model Functional Invocation Capacities: This schematic representation delineates one potential procedural implementation trajectory through exemplar construction and parametric optimization methodologies, comprising three principal operational phases: (1) Functional Entity Aggregation, wherein computational procedural constructs and their corresponding ontological descriptors are systematically accumulated and categorized; (2) Invocation Sample Architectural Development, potentially encompassing functional execution instantiations, parameter-deficient scenarios, and heterogeneous response configurational paradigms; and (3) Output Sample Structural Formulation, illustrating one methodological approach toward mapping execution resultant states to appropriate linguistic response generations. While other approaches exist (e.g., LlamaFactory [175]'s unified training framework), this pipeline demonstrates common components that practitioners might consider when implementing function calling capabilities.

The comprehensive resolution of aforementioned methodological impediments would facilitate enhanced linguistic model performance vis-à-vis complex user directive processing, precise functional invocation execution, and appropriate response generation while simultaneously preserving contextual relevance parameters and output consistency indicators. Through transcendence of current operational limitations, computational frameworks would attain superior capability regarding nuanced query interpretation, intricate workflow administration, and reliable output production across diverse interaction modalities. This augmented functional capacity would engender robust operational support spanning multitudinous application scenarios, encompassing rudimentary task automation implementations through sophisticated multi-iterative conversational exchanges, ultimately enhancing user satisfaction indices and systemic reliability coefficients. The resultant performance optimization would concurrently facilitate seamless integration capabilities with extant technological infrastructures while accommodating emergent utilization paradigms. Subsequent analytical sections shall elucidate detailed methodological approaches addressing these operational challenges through the prismatic perspectives of exemplar construction methodologies, parameter optimization protocols, and deployment architectures, examining specific procedural frameworks and implementation heuristics for each operational dimension.

3 ENHANCING LLMS TO HAVE THE ABILITY: SAMPLE CONSTRUCTION AND FINE-TUNING

In the previous section, we discussed the necessary steps for function calling. This chapter addresses how pre-trained LLMs under natural language settings can be endowed with function calling capabilities, as illustrated in Figure 2. As shown in Table 2, we present a systematic review of sample construction methods and fine-tuning strategies, ranging from function collection approaches to critical considerations in the training process.

Table 2. Sample Construction and Fine-Tuning Strategies for Function Calling in LLMs

Sample Construction & Fine-Tuning	Function Collection (§3.1)	**Manual Construction**: Human-crafted functions **LLM Generation**: Usage of LLMs for automated function creation **Web Mining**: Diverse function extraction from web
	Sample Construction (§3.2)	**Text Representation**: Toolformer [137], ToolGen [172] **Token Representation**: Toolformer [137], ToolGen [172] **Multi-turn Interaction**: GraphQL-RestBench [135], Hammer [78]
	Fine-tuning Strategies (§3.3)	**SFT**: ToolGen [172], RAIT [136], Nye et al. [103], Andor et al. [7], Liu et al. [79], Allamanis et al. [6], Barone et al. [11], Liu et al. [84], Liu et al. [82] **PEFT**: GPT4Tools [199], CITI [48], Toolformer [137], Li et al. [69], Wei et al. [181] **RL & RLHF**: WebGPT [98], TaskMatrix.AI [75], MADAC [73], GopherCite [126], Kojima et al. [63], DPO [127], Christiano et al. [25], Manduzio et al. [89]
	Critical Emphasis (§3.4)	**Data Quality**: Focus on data diversity and verification rather than volume **Model Scaling**: Larger models show better function calling capabilities **Capability Balance**: Need to maintain general abilities while enhancing function calling

3.1 Function Collection

The initial step involves collecting functions, including the function object—abstract entities consisting of the function name, parameters, and execution logic—and their descriptions. Real-world functions are often disorganised, and descriptions can be chaotic. Aside from manual construction, function objects and their descriptions can also be generated using large-scale LLMs like GPT-4 [2][1],LlaMA 70B [164] or Qwen [10, 195]. However, our experience suggests that such generated functions often lack diversity. Alternatively, data mining techniques can extract diverse function objects from the web, with descriptions supplemented by powerful LLMs if missing. By assembling a sufficient number of functions, the model can effectively grasp the timing of function triggers, thereby addressing **Challenge 2.1**.

3.2 Function Calling Sample Construction

We can construct sample mapping queries to outputs with a sufficient collection of functions. Real-world function calls involve returning the corresponding function and parameters, awaiting execution by the system and server. In cases where user-provided parameters are insufficient, the model may generate queries asking for additional information. Samples must, therefore, cater to these scenarios by including a variety of input-output pairs. Only when provided with a sufficient number of samples containing inadequate parameters, can the model learn when to request missing parameters from users, thereby partially addressing **Challenge 2.2**. Functions can be represented either as text or tokens in the model. Text representation provides more flexibility and semantic information but requires more token space, while token representation (as adopted by Toolformer [137] and ToolGen [172]) is more compact and computationally efficient but may have limited expressiveness. These two approaches can be combined - using token representation during training for efficiency while converting to text format during inference for better interpretability and interaction. For instance, both Toolformer and ToolGen encode functions as special tokens during training, but can present them in natural language format during interaction with users.**Challenge 3.1**.

Inputs typically consist of function candidates and a natural text query generated by robust LLMs based on original functions and descriptions. Outputs are crafted to match the varying requirements of real-life function-calling scenarios. For instance, in situations where function calls are not necessary, it is essential to construct examples to guide the model in understanding when to rely on its capabilities to respond. This approach could address **Challenge 2.2**. Moreover, samples can be constructed to simulate multi-turn interactions to enhance multi-turn function calling capabilities [20]. GraphQL-RestBench [135] further advances sequential function calling by introducing structured API schemas and response mapping, focusing on real-world REST API scenarios where functions have complex interdependencies.

To further improve robustness in function selection and address **Challenge 2.2**, recent work like Hammer [78] introduces specialized techniques such as function masking and dataset augmentation. This approach specifically targets naming convention sensitivity issues that often mislead models, while demonstrating state-of-the-art performance even in resource-constrained on-device deployment scenarios.

3.3 Fine-tuning Strategies

LLMs trained in natural language can perform function calls by designing prompts to enable this capability, as discussed in other sections. However, this approach is often insufficient because the model's reasoning and acting abilities depend entirely on the provided prompts, and the model itself does not self-optimize or adapt to new environments. Therefore, additional tuning is essential, mainly when new behaviors are challenging to learn and require more than just a few examples. Before fine-tuning LLMs, constructed function calling samples and response samples can undergo a uniform format cleaning process according to the specific usage, which typically includes a series of cleaning strategies, such as data deduplication [6], syntax normalization [11], format standardization [84], and error correction [82]. . Subsequently, some LLM fine-tuning strategies such as SFT (full-model fine-tuning), PEFT (parameter-efficient fine-tuning), and RLHF (reinforcement learning with human feedback) can be employed to optimize model performance via constructed and cleaned samples.

Specifically, parameter θ^* is learned by maximizing the likelihood of the output samples corresponding to the dataset constructed, as discussed in previous sections. The objective function calculates the expected value of the product of action probabilities for all queries q_i and their corresponding outputs a_i^* within the dataset $\mathcal{D}$, given the auxiliary cues $x_{i,t}$ and the conditions of the queries. Specifically, the objective function is defined as $\theta^* = \arg\max_{\theta} \quad \mathbb{E}_{(q_i,a_i^*)\in\mathcal{D}}\left[p_\theta(a_i^* \mid x_i, q_i)\right]$, where $p_\theta(a_i^* \mid x_i, q_i)$ represents the probability of output given the query and auxiliary cues. Beyond direct imitation learning, additional information and tasks can be integrated to enhance the training process. For example, Nye et al. [103] use execution traces as a form of supervision to improve reasoning abilities. In contrast, Andor et al. [7] apply heuristics to gather supervised data, facilitating the fine-tuning of language models. RAIT [136] combines instruction-tuning of code small language models with retrieval-augmented tool usage, enabling systematic problem-solving in process engineering. ToolGen [172] injects tool information through fine-tuning LLMs with tool descriptions as inputs and their corresponding tokens as outputs, directly incorporating tool knowledge into model parameters. Additionally, the integration of LoRA can further refine these approaches. For example, GPT4Tools [199] advance the capabilities of open-source LLMs by incorporating LoRA optimization techniques into fine-tuning, utilizing datasets composed of tool usage instructions produced by ChatGPT. CITI [48] proposes a novel Mixture-of-LoRA adapter to important and selectively fine-tunes unimportant components, enabling LLMs to enhance their tool-utilizing capabilities while preserving general performance across multiple tasks.

Reinforcement Learning (RL) in function calling enables LLMs to interact with external tools or APIs within a task directly, enhancing their ability to perform specific functions based on given inputs. For instance, Toolformer [137] utilizes RL-based techniques to bootstrap examples of tool use, improving the language model's effectiveness in function calling. Reinforcement Learning from Human Feedback (RLHF) [2, 25] further enhances this approach by aligning the language model more closely with complex human preferences and values that are challenging to capture with hardcoded reward functions. This proves particularly useful in function-calling scenarios [69] where the model interacts with external systems or tools to perform tasks such as computation, information retrieval, or navigating web interfaces. For example, WebGPT [98] employs a text-based browser to answer questions using internet searches, optimizing browsing actions and answer quality through RLHF. Similarly, GopherCite [126] is fine-tuned with RLHF to cite supporting evidence when answering questions and to abstain when unsure. TaskMatrix.AI [75] leverages RLHF to incorporate human feedback dynamically, enhancing the model's ability to navigate and coordinate the use of multiple tools and APIs. Recent work [89] proposes an efficient framework for training smaller models in function calling capabilities, using larger models to generate training data through step-by-step reasoning chains [63, 181] and Direct Preference Optimization (DPO) [127]. The safe multi-agent reinforcement learning framework proposed by MADAC [73], which ensures state-wise safety constraints while achieving optimal performance, could potentially provide insights for developing safe function calling systems. This approach demonstrates that focused training on specific reasoning tasks can achieve strong performance even with reduced model sizes, addressing practical deployment constraints.

In addition to their direct application [79], RL methods are crucial for repairing bad cases in real-world function-calling scenarios within industrial settings. There is a perceptual discrepancy between LLMs and human interpretation of queries. For example, when users search for 'flights from Los Angeles to Hangzhou on the evening of August 30th,' they might also consider flights in the early hours of August 31st as 'evening' flights. However, if an LLM strictly interprets the query, it may miss these flights. Relying solely on explicit data to resolve these discrepancies is impractical, as training LLMs to encompass all potential user intents is unfeasible. Furthermore, it is difficult to capture complex intentions through simple inputs due to the limitations in prompt length. Therefore, adjusting based on human feedback becomes crucial. Reinforcement Learning from Human Feedback (RLHF) is essential for function-calling learning because it allows for direct learning from human feedback, correcting errors, and refining system responses. This technology is critical for addressing habitual mistakes, misentered parameters, and even hallucinations in user inputs.

Fine-tuning strategies effectively address several key challenges. For **Challenge 3.1** and **Challenge 3.2**, approaches like ToolGen [172] demonstrate that directly incorporating tool knowledge into model parameters through fine-tuning can significantly reduce parameter errors and hallucination. To address **Challenge 2.2**, methods like GPT4Tools [199] and CITI [48] show that LoRA-based selective fine-tuning can effectively reduce unnecessary function invocations while maintaining model performance. Additionally, for **Challenge 4.1** and **Challenge 4.4**, RLHF approaches demonstrated by WebGPT [98] and MADAC [73] provide effective frameworks for handling execution failures and improving result accuracy through human feedback.

3.4 Critical Emphasis

Based on practical implementations, we emphasize that data quality (and variety) plays a more crucial role than data quantity in both data construction and fine-tuning phases, given the intricate nature of function calling tasks. To validate this hypothesis, we conducted a simple experiment: we selected a subset of 1,000 samples from the BFCL [112, 113] training dataset for supervised fine-tuning using LLaMA Factory [175], and used BFCL's Abstract Syntax Tree (AST) test category for evaluation. We chose AST tests as they enable offline evaluation of function calling capabilities without requiring external API access. We applied LoRA fine-tuning with a rank of 8 and a learning rate of 3e-4, training for 3 epochs. We specifically chose Qwen1.5-7B-Instruct [10] as our base model to ensure experimental integrity, as this older version predates the release of the Gorilla dataset, thus eliminating potential data contamination concerns. To investigate the minimal data requirements for achieving satisfactory function calling capabilities, we conducted experiments with varying training sample sizes. As shown in Figure 3, our experimental results demonstrate that model performance quickly reaches a plateau after processing approximately 400 training samples. This early plateau phenomenon suggests two important insights: First, the function calling ability can be effectively learned with a relatively small amount of high-quality data, indicating that the model can quickly grasp the underlying patterns of API usage and parameter

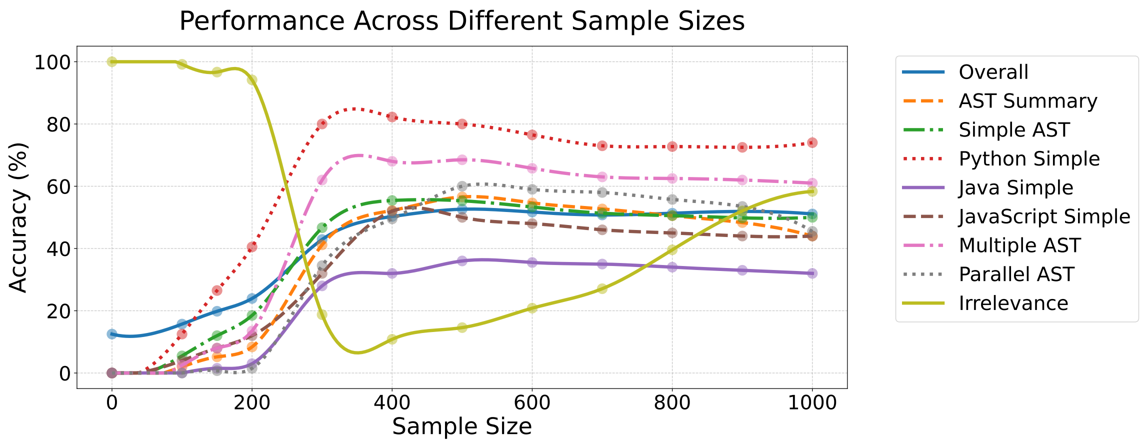

Fig. 3. Performance comparison of different models across various sample sizes. The results show that Python Simple achieves the best performance with a peak accuracy of 82.25% at 400 samples, while Multiple AST and AST Summary also demonstrate strong performance above 60% accuracy.

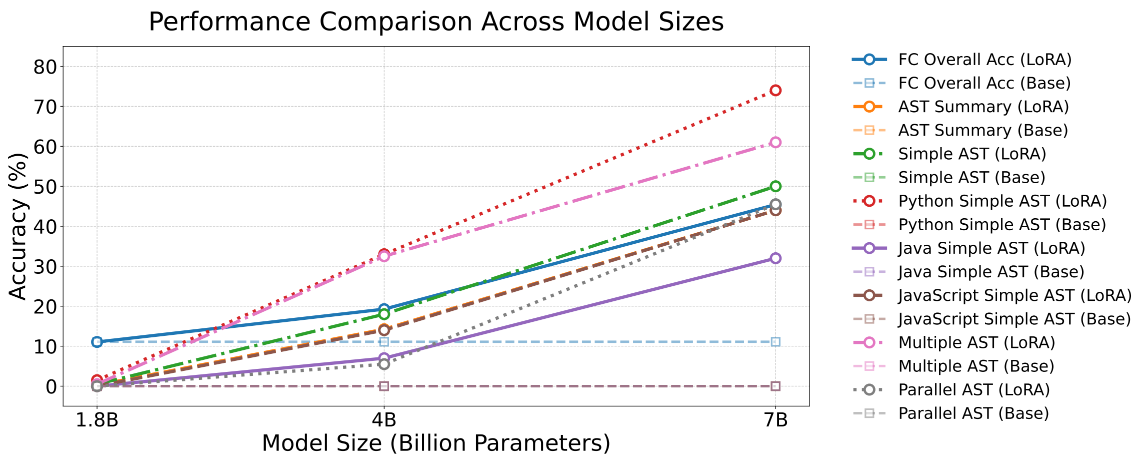

Fig. 4. Performance trends across different model sizes. Base models show near-zero function calling capabilities across all scales, indicating this ability is unlikely to emerge naturally during pre-training. After LoRA fine-tuning, performance exhibits clear scaling law characteristics, with particularly significant improvements from 4B to 7B. This suggests that function calling capabilities need to be explicitly introduced during both pre-training and fine-tuning stages and benefit substantially from larger model scales.

formatting; Second, simply increasing the training data volume beyond this point yields diminishing returns, which emphasizes the importance of data quality over quantity in function calling tasks. This finding has significant practical implications for efficient model training and resource utilization in real-world applications.

As shown in Figure 4, we conducted another experiment to investigate the impact of model size on function calling capabilities. We selected three Qwen models of different sizes (1.8B, 4B, and 7B) and evaluated their performance on BFCL's AST tests, both with and without fine-tuning. Our experiments reveal two key findings: First, base models without fine-tuning demonstrate minimal function calling capabilities, suggesting this skill rarely emerges naturally during pre-training. Secondarily, the performance trajectories subsequent to Low-Rank Adaptation parametric optimization methodologies manifest characteristic computational scaling law phenomenology, with particularly significant efficacy

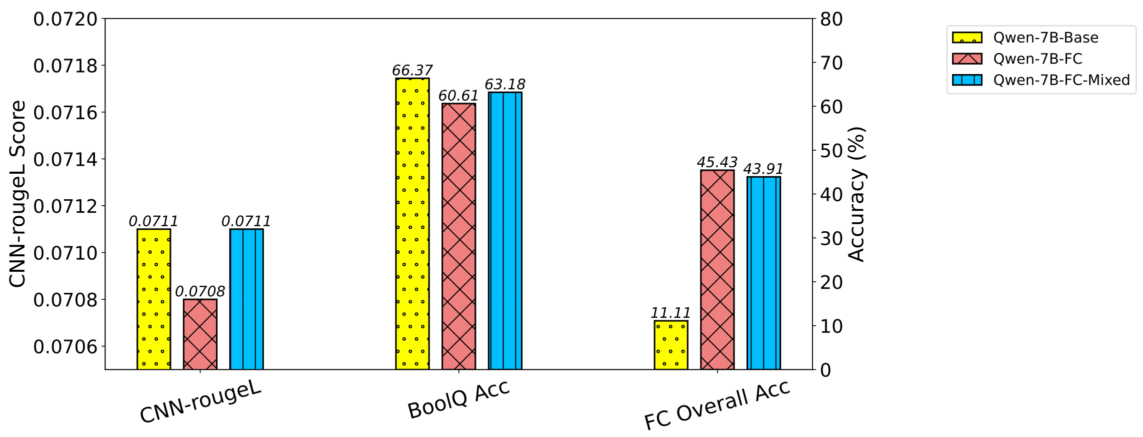

Fig. 5. The potential seesaw effect across different language capabilities. After fine-tuning Qwen-7B with function calling data, we observe a substantial improvement in function calling accuracy (FC Overall Acc increases from 11.11% to 45.43%). Meanwhile, the model shows varying performance changes in other capabilities: a decrease in question-answering ability (BoolQ accuracy drops from 66.37% to 60.61%) and relatively stable performance in news generation (CNN-rougeL maintains around 0.071). This pattern might suggest a seesaw effect between specialized function calling capability and general language abilities, particularly in question-answering tasks, though more systematic studies would be needed to confirm this hypothesis.

amplifications observed at the 7B parameter quantification threshold, indicative of enhanced functional invocation capability acquisition correlating positively with architectural volumetric expansions. This scaling pattern topology reveals critical epistemological insights regarding model dimensionality requirements for functional invocation operational tasks. Whereas computational frameworks exceeding 7B parametric quantities demonstrate promising performance enhancement coefficients, reduced-scale architectures (sub-7B configurations) exhibit persistent inadequacy in attaining comparable operational efficacy despite substantial training corpus augmentation. This empirical observation suggests that conventional optimization methodologies may prove insufficient for reduced-scale models to acquire robust functional invocation capabilities. For more resource-efficient deployment within computationally-constrained operational environments, subsequent investigative endeavors should explore enhanced methodological approaches for reduced-scale architectures, including externalized memory mechanisms to compensate for inherent capacity limitations, specialized training paradigms such as incremental complexity progression methodologies, or architectural reconfiguration strategies specifically engineered for functional invocation operational contexts. This empirical finding underscores the necessity of optimizing equilibrium between computational efficiency metrics and functional invocation capability coefficients, particularly within deployment scenarios where expanded-scale architectural implementations may present practical implementation constraints.

In practice, intensive function call training tends to compromise the model's general capabilities, particularly in text generation. As demonstrated in our experiments, this trade-off between specialized function call abilities and general language capabilities presents a significant challenge. To investigate this phenomenon more systematically, we conducted experiments with Qwen-7B under three scenarios: the base model, function calling fine-tuned (FC), and mixed training with both function calling and natural language samples (Mixed). As illustrated in Figure 5, our evaluation using AST overall accuracy, CNN/Daily Mail dataset [140]'s RougeL score, and BoolQ accuracy [26] reveals an interesting pattern: while function calling fine-tuning substantially improves the AST performance, it appears to come at a cost to certain general language capabilities, particularly in question-answering tasks. The text generation ability, however, remains relatively stable across different training configurations, suggesting a complex relationship

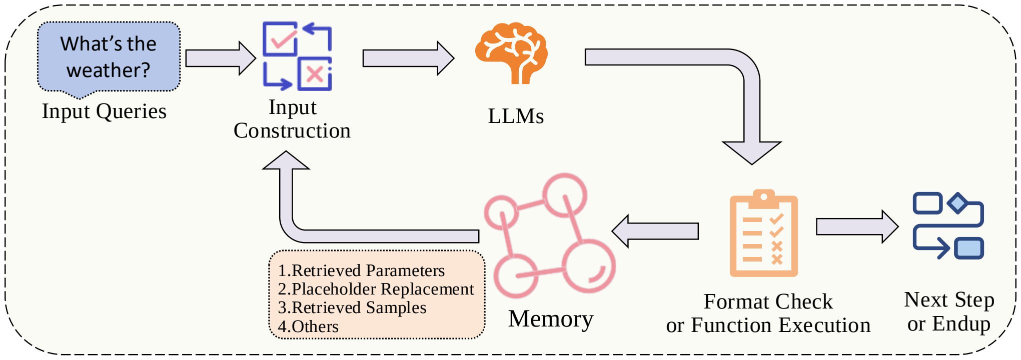

Fig. 6. A Typical Deployment of LLM for Function Calling Stages: The Flow through Input Construction, Memory Integration, and Output Format Validation (Function Execution). Note that actual implementations may vary in practice.

between different language capabilities. To address this challenge, we suggest incorporating domain-outlier data samples and natural language text [107] alongside function call examples during fine-tuning [162]. This mixed training approach attempts to maintain model versatility while developing function call competence, aligning with successful strategies that blend different types of training data [128]. While our initial results show promise in balancing these capabilities, further systematic studies would be needed to fully understand and optimize this relationship. Furthermore, concepts from lifelong learning and curriculum learning [174, 219] may offer promising directions for achieving a better balance between specialized and general capabilities, particularly crucial for practical applications where maintaining both sets of abilities is essential.

Some studies recommend sample augmentation [154] to improve tuning outcomes. APIGen [84] presents an automated pipeline for generating high-quality, verifiable function-calling datasets through a three-stage verification process, demonstrating that models trained on such carefully curated data can achieve superior performance even with relatively small parameter counts compared to larger models like GPT-4. ToolACE [82] presents an innovative self-evolving data synthesis pipeline that leverages multi-agent interactions and dual-layer verification to generate high-quality function-calling training data, enabling 8B parameter models to achieve GPT-4-level performance on standardized benchmarks. These approaches and the strategies discussed form a comprehensive methodology for tuning LLMs to handle sophisticated function-calling tasks effectively.

What's more, the ultimate goal for large models is to generate appropriate natural language text responses based on user queries. Data for mapping function return results to suitable natural language responses is therefore necessary.

4 IMPLEMENTING LLMS FOR FUNCTION CALLING: DEPLOYMENT AND INFERENCE

After constructing the sample and fine-tuning the large model, this section examines common deployment approaches that implement the three-stage pipeline (pre-call, on-call, post-call) outlined in Section 2. The deployment process for function-calling LLMs typically involves multiple inference steps aligned with these stages. While some implementations include an initial inference step for query understanding and decomposition using an intent model, this preprocessing phase continues to evolve in industrial applications and represents an area for future exploration. The core process of handling user queries and functions directly follows a well-defined workflow. As shown in Figure 6, a typical deployment involves input construction with the query and contextual information (pre-call), LLM-based function generation (on-call), and format check or function execution (post-call). The Memory component maintains essential context across these stages. This represents one common approach among various possible implementation patterns that practitioners

Table 3. Deployment and Inference Strategies for Function Calling in LLMs

Deployment & Inference	Task Planning (§4.1)	**Foundational Planning Mechanisms**: ReAct [203], ToolFormer [138], Reverse Chain [218], AVATAR [184], DEPS [178], LLM-MCTS [222], MACT [226], TACO [88], PAE [227], SCIAGENT [183], Agent Laboratory [139] **GUI-based Approaches**: AppAgent [202], OS-ATLAS [187], AndroidLab [194], Ponder [177], OS-Genesis [159] **System Optimizations**: Orca [93, 95], Memgpt [108], AIOS-Agent [41], SpecInfer [91], PEOA [153], LLM-Tool Compiler [150] **Error**: LLM-Planner [151], ToolChain* [229], TPTU [134], Buckets [23], AMOR [42] **Tree-based**: ControlLLM [85], PLUTO [50], Toolink [116], TPTU-v2 [64], α-UMi [145] **Adaptive**: COA [38], DEER [43], SOAY [176], ProgPrompt [149], AutoTOD [217], MATMCD [143], CC-PP [49], AVT [198], K-agents [18], Agent-Pro [217], Inner [83]
	Prompt Construction (§4.2)	**Few-shot Integration**: Example demonstrations [154], Four-shot prompting **Context Management**: Function definitions, Docstrings, Chain-of-thought **Query-based Retrieval**: Ask-when-Needed [173], Interactive refinement
	Function Generation (§4.3)	Grammar control [111, 166], Knowledge guidance [17] Attribution [231], Feedback [192], Dialogue refinement [59] Multi-agent coordination [46], Task proposal [227], Experience transfer [114]
	Function Mapping (§4.4)	**Resolution**: Rule-based [66, 81], Knowledge reasoning [214], LLM mapping [67] **Alignment**: Dictionary mapping [156], Semantic matching [121], normalization **Validation**: Parameter checking, Value enumeration, Permission management
	Response Generation (§4.5)	**Initial Generation**: Placeholder results [47, 110, 137], Function unpredictability [203] **Templates**: Structure format [113, 120], Formatting [68], Signatures [154] **Review**: Validation [120, 152], Agent correction [55, 147], Feedback [51, 99, 190] **RAG**: Example retrieval [40], System mapping [19, 76, 100, 129, 133, 154, 209]
	Memory Scheme (§4.6)	**Memory Structure**: Hierarchical structure and storage [225], Task-related symbolic memory [201], Three-layered memory architecture [71], Persistent memory stream [74] **Memory Management**: Self-controlled memory mechanism [74], Memory control system [74, 94], Multi-agent experience storage [72] **Memory Retrieval**: Cross-conversation memory retrieval [225], LSH-based indexing mechanism [94], Similarity-based retrieval [223], Efficient memory access [80] **Memory Processing**: Thought-based memory storage [80], Trajectory-as-exemplar framework [223], State abstraction mechanism [223], Knowledge triplet [94]

have explored. As detailed in Table 3, we provide a comprehensive overview of deployment and inference strategies, covering aspects from task planning to memory schemes.

4.1 Task Planning and Compiler

4.1.1 ***Foundational Planning Mechanisms.*** Understanding user intent alone is insufficient for complex function calling tasks, necessitating sophisticated task planning approaches. Early works established fundamental planning capabilities through chain-of-thought prompts (ReAct [203]) and fine-tuning approaches (ToolFormer [138]), enabling LLMs to decompose problems and utilize APIs progressively. Reverse Chain [218] advanced this by introducing target-driven backward reasoning for more controlled multi-API planning without fine-tuning. Recent frameworks have enhanced planning capabilities through multi-component architectures and iterative refinement. Agent Laboratory [139] advances structured research planning through a multi-agent architecture where specialized agents (PhD, Postdoc, Professor) collaborate using function calling and iterative planning mechanisms - each agent executes targeted commands (e.g., SEARCH, DIALOGUE, EDIT) while engaging in reflective dialogue to decompose complex research tasks, similar to how AVATAR [184] leverages actor-comparator architecture but adapted specifically for scientific workflows. The system enables iterative refinement through stage-wise human feedback, analogous to DEPS's interactive planning approach, while maintaining research state tracking across literature review, experimentation and writing phases. DEPS [178] enables interactive planning through description-based decomposition and iterative refinement, while LLM-MCTS [222] combines Monte Carlo Tree Search with memory augmentation, leveraging LLMs as both a commonsense world model and a search heuristic. These planning approaches have been successfully adapted to diverse application scenarios.

For complex table analysis, MACT [226] implements iterative planning between planning and coding agents through action generation and execution. TACO [88] enhances multi-modal models through synthetic Chain-of-Thought-and-Action traces for complex task solving. PAE [227] improves zero-shot generalization through context-aware planning and evaluation beyond existing models. In specialized domains, SCIAGENT [183] enables direct scientific reasoning through tool-augmented planning across multiple domains, while Ning et al. [101] combine Code Property Graphs with LLM-based semantic analysis for comprehensive agent code defect detection. To enable effective planning capabilities, FlowAgent [146] introduces Procedure Description Language (PDL) that bridges natural language flexibility with code-like precision, allowing LLM-based agents to adapt workflows while maintaining procedural control. These advances demonstrate the evolution from basic chain-of-thought approaches (ReAct [203], ToolFormer [138]) to sophisticated planning frameworks (AVATAR [184], DEPS [178], LLM-MCTS [222]) capable of handling diverse tasks through structured decomposition, iterative refinement, and domain-specific adaptations (MACT [226], TACO [88], SCIAGENT [183]).

4.1.2 ***GUI-based Approaches.*** Recent work has explored GUI-based approaches to enhance function calling capabilities. AppAgent [202] proposes a two-stage training paradigm combining GUI grounding pre-training and action fine-tuning to enable LLMs to understand and interact with mobile applications. OS-ATLAS [187] further extends this by introducing a foundation action model for generalist GUI agents, unifying the action space across different platforms. These approaches demonstrate the potential of combining API and GUI-based methods for more comprehensive task execution capabilities. AndroidLab [194] provides a systematic benchmark framework for evaluating such autonomous agents in real-world mobile environments. Ponder & Press [177] proposes a divide-and-conquer visual GUI agent framework that uses only visual input, featuring a two-stage planning strategy with instruction interpretation and element localization to enable direct and general computer control. OS-Genesis [159] proposes an interaction-driven GUI agent trajectory construction framework that generates training data by exploring environments first, then deriving tasks retrospectively, without human supervision.

4.1.3 ***System-level Optimizations.*** From a system perspective, researchers have explored optimizing LLM execution through compiler and operating system innovations. At the compiler level, Orca [93, 95] proposes a specialized approach to optimize LLM inference through instruction scheduling and memory management, introducing techniques like operation fusion and memory layout optimization to reduce inference latency. Operating system optimizations have also shown promising results. Memgpt [108] presents an operating system design specifically tailored for LLM workloads, providing efficient resource management and scheduling mechanisms. AIOS-Agent ecosystem [41] envisions a revolutionary architecture where LLM serves as the operating system kernel while diverse AI agents function as applications, enabling natural language programming and democratizing software development. These system-level optimizations are crucial for enabling efficient on-device LLM deployment and execution, particularly in resource-constrained environments where memory and computational efficiency are paramount. Recent system optimizations further explore specialized architectures. SpecInfer [91] introduces speculative inference to enhance serving efficiency through parallel execution. The PEOA [153] introduces a Large Language Model Operating System (LLM OS) with a modular architecture, where a meta-agent orchestrates an action generator and specialized expert models to break down and solve complex chemical engineering problems, utilizing property graph-based knowledge modelling and teacher-student transfer learning with GPT-4 for improved tool integration and problem-solving capabilities, while managing system resources and tool scheduling in a manner similar to traditional operating systems. LLM-Tool Compiler [150] fuses similar tool operations into unified tasks at runtime, achieving up to 4x improvement in parallel execution while reducing API costs in function calling systems. These works demonstrate the potential of system-level optimizations in improving LLM performance across different deployment scenarios.

4.1.4 ***Robust Planning through Error Handling.***

Error handling and recovery mechanisms have been explored to enhance planning reliability. LLM-Planner [151] introduces environmental feedback for plan regeneration during execution failures, while ToolChain* [229] employs decision trees to systematically manage API calls. TPTU [134] and Attention Buckets [23] focus on reducing information loss through structured frameworks and parallel operations. AMOR [42] presents a finite state machine (FSM)-based framework for function calling that allows autonomous execution and transitions over disentangled modules, enabling process-level human feedback to improve reasoning capabilities through a two-stage fine-tuning approach (warm-up and adaptation).

4.1.5 ***Tree-based Decision Making.***

Tree-structured approaches offer systematic solution exploration. Control-LLM [85] implements Tree of Thoughts with depth-first search on tool graphs. PLUTO [50] uses autoregressive planning with hypothesis trees, while Toolink [116] and TPTU-v2 [64] leverage hierarchical task decomposition. α-UMi [145] extends this through specialized planning-oriented fine-tuning.

4.1.6 ***Adaptive Planning Strategies.***

Recent works advance flexible planning and coordination through various approaches. In terms of adaptive planning, COA [38] employs abstract reasoning chains that adapt to domain knowledge, while DEER [43] enhances generalization through dynamic tool sampling. SOAY [176] and ProgPrompt [149] further this adaptability by generating executable code structures tailored to specific execution environments. Building on adaptive planning, several frameworks demonstrate effective task decomposition and multi-agent coordination. MATMCD [143] introduces a multi-agent framework where data augmentation and causal constraint agents collaborate for multi-modal causal discovery. Similarly, AVT [198] decomposes video processing tasks between captioning and arrangement agents, while K-agents [18] coordinates translation and inspection agents for experimental procedures using state machines. Advanced coordination mechanisms are explored in several works. CC-PP [49] employs a two-stage approach combining path heuristics with greedy best-first search for multi-agent coordination under communication constraints. AutoTOD [217] demonstrates effective task planning through instruction-following models, while Agent-Pro [217] introduces dynamic belief management and policy-level reflection. The Inner Thoughts framework [83] enhances multi-party coordination by enabling proactive participation through continuous thought generation and evaluation.

Task planning and system optimization address several key challenges in function calling. For **Challenge 1.1** complex task decomposition and **Challenge 1.2** execution planning, foundational planning mechanisms like chain-of-thought and multi-component architectures provide systematic approaches for task breakdown and execution. For **Challenge 2.1** cross-modal interaction and **Challenge 2.2** system efficiency, GUI-based approaches and system-level optimizations enable effective human-computer interaction and improved computational performance. Additionally, for **Challenge 3.1** error handling and **Challenge 3.2** robustness, various frameworks introduce mechanisms for execution reliability and recovery strategies.

4.2 Prompt Construction Strategies

For function calling tasks, effective prompt construction requires careful integration of few-shot examples, context, and query-based retrieval. NexusRaven [154] demonstrates that retrieving demonstrations from existing query-response pairs, using approximately 16 examples per API function with four-shot prompting, significantly improves function calling success rates. The prompt context must include function definitions, docstrings, capability descriptions, and chain-of-thought components that explain argument derivation and function selection logic. Hard-negative examples of similar but incorrect functions help improve the model's discrimination ability. The construction process leverages demonstration retrieval by scanning existing query-response pairs while maintaining a growing corpus of past use cases that enables system personalisation through live interactions. This approach incorporates multi-step refinement, from mining raw function calls to generating natural language queries with chain-of-thought reasoning. While these methods focus on improving function calling through demonstration and refinement, the Ask-when-Needed prompting strategy [173] enhances LLMs' ability to handle unclear instructions by encouraging proactive question-asking before API calls, achieving significant improvements in accuracy while maintaining efficiency when paired with GPT-4. These prompt engineering strategies effectively address **Challenge 2.1**, **Challenge 2.2**, and **Challenge 3.2** by improving function triggering accuracy and reducing hallucination through better-structured prompts and examples.

4.3 Function Generation (or Selection) Strategies

Once the prompt is constructed, the LLM proceeds to generate functions. A critical step in this process involves the controlled or restricted generation of outputs. Control over function generation in LLM function calls can be significantly enhanced through the use of context-free grammar (CFG) [111, 166]. By adopting a CFG-based approach, the model's output can be restricted to syntactically valid sequences, effectively constraining the token space to prevent the production of invalid tokens. This methodological implementation operationalizes a filtration mechanism that orchestrates the selective procedural determination during generative processes through syntactical constraint application, ensuring rigorous conformity to grammatical structural specifications delineated within the Context-Free Grammar framework. As empirically substantiated through application deployment scenarios, this procedural architecture not only expedites integration protocols of computational linguistic outputs through syntactical anomaly reduction but concurrently enhances reliability coefficients of model-generated content, thereby preserving elevated qualitative generation standards within predetermined grammatical constraint parameters.

TOOL-ED [17] methodological framework augments affective conversational content generation through reconceptualization of epistemological repositories such as COMET as invocable procedural utilities, facilitating linguistic model implementation of flexible accessibility protocols to externalized emotional knowledge constructs through pedagogical-acquisitional transfer learning paradigms, consequently achieving superior performance metrics in affective response generation while simultaneously circumventing superfluous knowledge incorporation phenomena.

VisionMask [231] proposes an attribution-centric architectural paradigm that identifies critical visual-spatial regions within optical input modalities catalyzing functional invocation processes through autonomously-supervised educational methodologies, enabling enhanced comprehension and validation capabilities regarding computational decisional processes. TR-Feedback [192] proposes an iterative feedback framework that leverages LLMs to enhance tool retrieval performance through comprehension, assessment, and instruction refinement while establishing both in-domain and out-of-domain benchmarks for tool retrieval evaluation, significantly improving tool retrieval accuracy. APEC-Travel [59] proposes an interleaved dialogue framework that effectively extracts personalized user preferences through multi-round interactions, demonstrating how to achieve accurate, proactive, efficient, and credible function calling via streaming processing. IBSEN [46] proposes a director-actor agent collaboration framework for generating controlled drama scripts, demonstrating an effective function generation strategy through multi-agent coordination and dynamic adaptation. PAE (Proposer-Agent-Evaluator) [227] demonstrates significant improvements in accuracy by introducing a three-component framework that combines context-aware task proposal, chain-of-thought agent execution, and image-based outcome evaluation, achieving over 30% relative improvement in zero-shot generalization to unseen

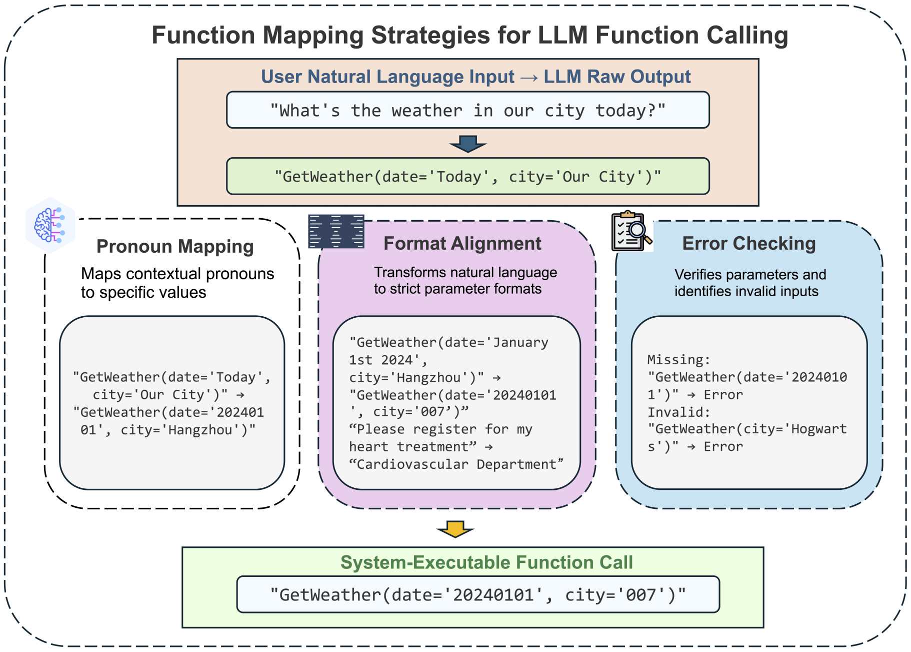
Function Mapping Strategies for LLM Function Calling
User Natural Language Input → LLM Raw Output
"What's the weather in our city today?"
"GetWeather(date='Today', city='Our City')"
Pronoun Mapping
Maps contextual pronouns to specific values
"GetWeather(date='Today', city='Our City')" → "GetWeather(date='20240101', city='Hangzhou')"
Format Alignment
Transforms natural language to strict parameter formats
"GetWeather(date='January 1st 2024', city='Hangzhou')" → "GetWeather(date='20240101', city='007')"
"Please register for my heart treatment" → "Cardiovascular Department"
Error Checking
Verifies parameters and identifies invalid inputs
Missing: "GetWeather(date='20240101')" → Error
Invalid: "GetWeather(city='Hogwarts')" → Error
System-Executable Function Call
"GetWeather(date='20240101', city='007')"

Fig. 7. Function mapping strategies in LLM function calling, illustrating the transformation process from natural language input to system-executable function calls through pronoun mapping, format alignment, and error checking.

tasks and websites even when using weaker models for task proposal and evaluation. Experiential Co-Learning [114] proposes an experience-based collaborative framework for software code generation, demonstrating improved function generation through agent cooperation and experience transfer

These approaches can improve the accuracy of LLMs in function calls, which demand high precision, thereby addressing **Challenge 3.2** and **Challenge 3.4**.

4.4 Function Mapping Strategies

Function mapping plays a crucial role in deploying function calling, primarily responsible for transforming model outputs at the semantic level into executable commands in the physical space. Moreover, as shown in Fig. 7, function mapping involves Pronoun Mapping, Format Alignment, and Error Checking.

4.4.1 ***Mapping of Pronouns.*** First, it is necessary to map some pronouns. For instance, when querying "the weather in our city today" the model output "GetWeather(date='Today', city='Our City')" can be recognized and converted to "GetWeather(date='20240101', city='Hangzhou')" This can be achieved through predefined human-designed rules (by constructing and maintaining necessary information mapping dictionaries) or by a small language model. Proper construction of the LLM data itself can also address this issue, but robust backups for industrial scenarios are very important. These conversions are typically achieved through simple mappings via human-designed rules [66, 81], knowledge structures reasoning [214], resolving and mapping via small language models [67], UI-guided token selection and interleaved streaming [121], or directly adding some specific tuning samples to finetune and enhance the LLM's direct resolving capabilities, thus addressing **Challenge 3.3** and **Challenge 4.3** to some extent.

4.4.2 ***Strict Format Alignment Mapping.*** Additionally, there is often a large gap between the user's natural language input and the strict requirements of function parameters (which change according to the function). For example, when querying the weather in "Hangzhou" on "January 1st, 2024," the model output "GetWeather(date='January 1st 2024', city='Hangzhou')" is converted to "GetWeather(date='20240101', city='007')" to meet actual system requirements. Or, when a user inputs "Please register for my heart treatment" at a hospital, it needs to be mapped to "Cardiovascular Department" This mapping can be maintained in the service through an information mapping dictionary, or by allowing the LLM to perform a first-step alignment through LLM CoT, or by training a small language classifier with semantic features to match function parameters. Syllabus [156] introduces a portable curriculum learning library that provides unified APIs and format alignment mechanisms for training reinforcement learning agents across different environments and frameworks. These approaches also address **Challenge 4.3** to some extent.

4.4.3 ***Identification of Invalid Inputs.*** In addition, the mapping stage also needs to verify that all required parameters are included and assess the accuracy of enumerated values. This effectively addresses issues of missing or invalid parameters and mitigates the impact of **Challenge 3.1** at the system level. For example, if a model returns a query "GetWeather(date='20240101')" missing city information, the parameter checker will identify the missing city parameter and trigger a default response configured by operations, which is then returned to the user. Or if a query requests tomorrow's weather in "Hogwarts" with the output "GetWeather(date='20240101', city='Hogwarts')" and "Hogwarts" is not recognized in the city code table, the checker will flag it as an invalid value and trigger a default response for unsupported values, which is then communicated to the user.

Moreover, function mapping could also involve filling in some parameters invisible to the LLM, such as fields related to permissions and UUIDs, which are supplemented during the backend processing stage to ensure the request's integrity and security. Function mapping fundamentally involves utilizing system-level rules and algorithms to ensure the robustness of the function-calling process and enhance the user experience in a way that is both reliable and intuitive. Effective function mapping addresses **Challenge 4.3** and can mitigate the impact of **Challenge 3.1** at the system level.

4.5 Response Generation Strategies

The simplest method for response generation involves initially using a model to generate placeholder results [47, 110, 137], which are then replaced with the outputs from an API call (either through direct matching or rewriting with a larger model). This approach ensures the model's planning and intent understanding are executed end-to-end. However, this method makes the function calls and their outputs unpredictable [203], and the execution of functions may lead to unexpected results.

4.5.1 ***Templates.*** Templates play a crucial role in structuring function calling outputs. Several works have demonstrated that structured templates significantly improve function calling accuracy and parameter selection [113, 120]. The template approach has proven especially valuable in API interaction scenarios where precise parameter formatting and type checking are crucial [68]. Studies indicate that well-designed templates can help models better understand function signatures and generate more accurate API calls [154]. These template-based approaches effectively address **Challenge 3.1** and **Challenge 4.3**.

4.5.2 ***Review.*** Response review mechanisms ensure reliable function execution through systematic validation. Parameter boundary checking and type validation during review phase have proven effective in preventing common function calling errors [120]. Beyond simple validation, interactive review and correction approaches have emerged - Song et al. implemented a schema-based parameter checking and response parsing system [152], while Shi et al. proposed a cooperative multi-agent framework that enables specialized agents to dynamically review and correct each other's actions [147]. A comprehensive evaluation by Jacovi et al. demonstrated that careful review mechanisms are crucial for tool-assisted systems, though they also found that existing review approaches still struggle with effective tool integration and validation [55]. Nathani et al. further advanced this field by introducing a multi-aspect feedback framework that integrates multiple specialized review modules to address different types of errors, showing significant improvements in reasoning accuracy [99]. Xu et al. proposed using compression and selective augmentation during review to make the process more efficient while maintaining effectiveness [190]. QueryAgent [51] proposes an environmental feedback-based framework for reliable and efficient function generation, introducing stepwise self-correction and targeted error guidance. These mechanisms are crucial for ensuring successful execution in complex API interactions. These review mechanisms effectively address **Challenge 3.1**, **Challenge 4.1**, and **Challenge 4.4**.

4.5.3 ***RAG.*** Retrieval-Augmented Generation (RAG) enhances function calling accuracy by leveraging existing examples [40]. By incorporating previously successful function calls, RAG systems can better map user queries to appropriate API calls [19, 76, 100, 129, 133, 154, 209]. Nguyen et al. propose SFR-RAG [100], a contextually faithful LLM specifically optimized for retrieval-augmented generation, introducing a standardized evaluation framework and demonstrating strong performance in contextual understanding and citation abilities with significantly fewer parameters than existing solutions. Chan et al. [19] evaluate RAG for API integration by comparing semantic search approaches using different embedding models and analyzing the limitations of retrieval-based methods compared to fine-tuning based solutions. The NexusRaven [154] shows that carefully curated demonstration sets can significantly improve parameter selection and function matching. These enhancements help maintain a high level of clarity and consistency in responses, which is crucial for applications requiring high reliability and ease of use. These strategies not only optimize the interaction between users and AI systems but also contribute to creating a more efficient and user-friendly experience. These approaches can effectively address **Challenge 3.4**, **Challenge 4.1** and **Challenge 4.2**.

4.6 Memory Scheme Strategy

Implementing an effective memory scheme in multi-turn dialogue situations is crucial, especially when it comes to handling function calls and parameter extraction. Given the empirical observation of suboptimal contextual continuity preservation within multi-iterative conversational frameworks, potentially resulting in deficient parameter extraction methodologies, we propose an architectural implementation strategy that discretely maintains functional parameter extraction resultants across the most recent K conversational iterations. By way of illustrative exemplification, during initial conversational engagement, when a user solicits meteorological information regarding "the weather in Hangzhou on 1st January 2024," the computational framework extracts GetWeather(date="20240101", city="Hangzhou")" and systematically preserves this parametric configuration within temporary computational storage. During subsequent conversational iteration, upon user inquiry regarding What about Xi'an?", the system extracts exclusively city=Xi'an" from the current linguistic input. At this operational juncture, the system reappropriates the temporal parameter from preserved memory architecture and executes GetWeather(date="tomorrow", city="Xi'an")". This methodological approach potentially incorporates memory attenuation mechanisms, wherein early conversational parametric configurations undergo systematic elimination once conversational iterations exceed predetermined threshold K.

The implementation of dynamic filtration and selective retention of contextual informational components predicated upon current interrogative specifications constitutes a critical operational dimension in enhancing dialogical qualitative metrics and relevance coefficients. This methodological paradigm effectively attenuates informational entropy derived from contextually irrelevant data while ensuring sustained computational focus on immediate conversational requirements. These optimization architectural implementations contribute significantly to experiential user satisfaction metrics and systemic operational efficacy in managing multi-iterative dialogical exchanges. To address these methodological impediments and further advance dialogical capabilities, investigative researchers have proposed various memory-augmented architectural paradigms. Among these theoretical frameworks, MemoryBank [225] presents an innovative memory-augmented planning architecture that enhances large linguistic computational models with extended temporal memory capabilities through three principal operational components: hierarchical memory storage infrastructure, Ebbinghaus-inspired memory updating algorithmic mechanisms, and cross-conversational memory retrieval methodologies. SCM [74] proposes a self-controlled memory-augmented planning framework that enables LLMs to maintain and utilize long-term memories through a memory stream, memory controller, and LLM agent architecture. TiM [80] proposes a memory-augmented planning framework that enables LLMs to store and recall thoughts rather than raw conversations, featuring pre-response recalling and post-response thinking to achieve consistent reasoning with efficient memory retrieval. RET-LLM [94] proposes a memory-augmented planning framework featuring a general read-write memory system that stores knowledge in triplets, manages memory through a controller, and enables efficient memory retrieval through LSH-based indexing. SYNAPSE [223] proposes a memory-augmented planning framework that uses trajectory-as-exemplar prompting, featuring state abstraction for memory efficiency, complete trajectories as exemplars for planning, and similarity-based memory retrieval for experience reuse. METAAGENTS [72] proposes a memory-augmented planning framework for collaborative generative agents, featuring perception for environment understanding, memory for experience storage, reasoning for planning and reflection, and execution for skill utilization. TradingGPT [71] proposes a memory-augmented multi-agent trading system that features a three-layered memory structure, inter-agent debate mechanism, and personalized trading characteristics to enhance financial decision-making through collaborative planning. DoraemonGPT [201] proposes a memory-augmented framework for dynamic scene understanding, featuring task-related symbolic memory, spatio-temporal and knowledge tools, and MCTS-based planning for exploring multiple solution paths. These memory management strategies and memory-augmented frameworks effectively address **Challenge 3.6** by enabling more coherent and context-aware interactions across extended conversations. Additionally, by maintaining accurate parameter history and context, these approaches help mitigate **Challenge 3.1** and **Challenge 3.3** through improved parameter extraction and pronoun resolution in multi-turn dialogues.

4.7 Function Call Latency Strategies

Function calls require low latency to perform efficiently, especially under the challenge of managing long contexts. Key-value (KV) cache techniques, commonly used to manage extensive contexts, can adversely affect performance when the context becomes overly lengthy, as they are not ideally suited for function calls. However, employing page attention mechanisms within the vLLM (very large language model) framework can effectively address these latency issues by optimizing how the model accesses and processes large contexts, significantly improving baseline LLM performance. vLLM [169] introduces continuous batching and PagedAttention for efficient serving of LLMs, achieving higher throughput while maintaining low latency. PagedAttention [65] presents a novel memory management mechanism that enables efficient attention computation through page-based memory organization, significantly reducing memory requirements for LLM serving. FlashAttention [28, 29, 141] presents an IO-aware attention algorithm that reduces memory access and increases training speed. SpecInfer [91] introduces speculative inference to enhance serving efficiency through parallel execution.

Due to the potential involvement of multiple functions in function calling (FC), system-level planning to optimize latency is essential. For instance, the LLM Compiler [61, 150] effectively plans and executes multiple function calls in parallel, organizing them using a Directed Acyclic Graph (DAG) to manage dependencies efficiently. This strategy not only streamlines the process but also significantly cuts down on latency, providing a more responsive system for handling intricate function call scenarios. Besides mitigating **Challenge 3.4**, the LLM Compiler also effectively addresses **Challenge 3.5** by managing the orchestration of sequential and parallel tasks seamlessly.

Several optimization techniques have proven effective for further advancements in reducing latency during function calls. Model pruning [87, 96, 158] and knowledge distillation [193, 197] demonstrate significant latency reductions while preserving function calling accuracy. Quantization methods [34, 167] could further optimize inference speeds in production environments. ThorV2 [14] demonstrates enhanced function calling capabilities with smaller models, achieving superior performance in CRM operations compared to larger commercial LLMs. TinyAgent [35] offers solutions through efficient model compression and optimization techniques, demonstrating that small language models can achieve GPT-4-Turbo-level performance while running entirely on local hardware like MacBooks. This marks a significant advancement in edge-based function calling capabilities. These approaches directly address **Challenge 3.4** by enhancing model efficiency in real-time applications.

5 LOOKING AHEAD: OPEN ISSUES AND DISCUSSIONS FOR FUNCTION CALLING IN LLMS

Despite the significant progress in function calling capabilities for LLMs, several critical open issues remain to be addressed. These issues span various aspects of function calling systems, from fundamental service-level issues to practical concerns about usability, optimization, and function isolation.

5.1 Open Issue 1: Service Issues of Function Calling

Establishing universally accepted standards for assessing function call quality across services remains challenging. Current evaluation metrics, as shown in **Appendix** Sec. A predominantly focuses on technical capabilities rather than service-oriented application requirements. These quantitative methodologies often prioritize computational efficiency but often inadequately address user experience and domain-specific implementation needs. LLMs often exhibit suboptimal performance characteristics, including elevated response latency and reduced throughput, especially when tool learning is integrated into reasoning frameworks. For instance, simple queries using LLM plugins can result in noticeable delays compared to traditional search engines. Reducing this latency is essential for maintaining service quality. In Agent Laboratory [139], this challenge is particularly pronounced as research processes require sequential tool usage across multiple stages.

Recent work by Wu et al. [186] has identified critical security vulnerabilities specific to function calling capabilities. Their study demonstrates that through a novel "jailbreak function" attack method, attackers can achieve over 90% success rates in bypassing safety measures across major models, including GPT-4 [104], Claude-3.5-Sonnet [8], and Gemini-1.5-pro [163].

Developing an integrated assessment framework that evaluates response time, accuracy, user satisfaction, and security measures is essential. Such a framework should incorporate multidimensional metrics that align technical performance with service quality and safety objectives.

5.2 Open Issue 2: Usability and Modification of Functions for Function Calling Applications

In practical applications, the effectiveness of function calls largely depends on the usability of callable functions, such as modifying existing functions. Identifying operational components suitable for computational utilization and addressing implementation impediments within specific scenarios constitutes a primary obstacle for optimized function invocation efficiency.

Challenges in function modification include ensuring data interoperability, circumventing architectural constraints, and addressing integration complexities between functional components and existing frameworks, which can potentially impede deployment flexibility and the efficacy of training for function invocation operations.

To mitigate technical barriers and economic factors, establishing standardized API modification frameworks becomes imperative, encompassing evaluation protocols, design specifications, validation methodologies, and deployment guidelines. Implementing modularized design paradigms can simplify integration protocols and enhance systemic flexibility.

Achieving equilibrium between economic resource allocation for API modifications and the resulting performance enhancements, while ensuring stability and effectiveness across diverse operational scenarios, remains an area that requires additional research.

5.3 Open Issue 3: Feedback Quality and Optimization of Function Calls

The efficacy of LLMs in acquiring and operationalizing human feedback remains insufficiently elucidated, particularly within scenarios involving complex or nuanced feedback mechanisms. Current implementations may inadequately utilize feedback due to complex processing sequences, potentially leading to error propagation and suboptimal learning outcomes.

Existing methodologies include Reinforcement Learning from Human Feedback (RLHF), wherein models undergo optimization utilizing reward signals derived from human preferences. Additionally, supervised optimization with annotated data repositories attempts to incorporate human input directly within training frameworks. However, these approaches often demonstrate inadequate performance with ambiguous or unstructured feedback, which is prevalent in authentic environments.

Further research is needed to develop methodologies for accurately quantifying and responding to human feedback, especially in ambiguous situations. Potential solutions include developing advanced natural language comprehension algorithms that can interpret underlying user intent, incorporating probabilistic modeling to manage uncertainty in feedback interpretation, and integrating interactive learning frameworks that enable clarification queries to resolve ambiguities.

5.4 Open Issue 4: Function Isolation and Post-Processing Strategies

Implementing appropriate isolation mechanisms and post-processing frameworks for distinct functional components is imperative to meet commercial requirements and regulatory compliance specifications. These implementations necessitate flexible design methodologies and elevated customization capabilities.

Examples include microservice architectures, wherein individual functional components are deployed as autonomous service entities with precisely delineated interfaces. This approach facilitates enhanced control over operational behavior and simplifies regulatory compliance by isolating sensitive processes. Additionally, middleware layers designed explicitly for post-processing tasks ensure that computational outputs conform to the necessary specifications.

A primary challenge is ensuring that isolation strategies and post-processing frameworks maintain effectiveness across heterogeneous implementation scenarios, while satisfying regulatory requirements and preserving service efficiency. Potential solutions might include adaptive algorithms that dynamically reconfigure API characteristics based on real-time analytics, policy-driven management systems that facilitate automated compliance mechanisms, and machine learning models for monitoring performance metrics to enable resource allocation adjustments.

6 CONCLUSION

This comprehensive survey has examined function calling in LLMs from an industrial perspective, systematically analyzing the challenges, solutions, and future directions in this rapidly evolving field. We have outlined a complete function calling pipeline consisting of three critical stages: pre-call, on-call, and post-call, and detailed the specific challenges encountered at each stage. The key challenges include intent recognition, function redundancy, parameter extraction, function hallucination, and multi-call procedures. Our review and discussion demonstrated that function calling significantly enhances LLMs' capabilities by enabling structured outputs and real-time interactions with external systems. As the field continues to evolve, we anticipate the development of more sophisticated techniques for context management, complex multi-function calls, and real-time parameter validation, leading to more powerful and practical AI systems that can better serve human needs while maintaining reliability and efficiency.

Additionally, detailed evaluations, benchmarking datasets, and analyses of industry products for LLM Function Calling can be found in **Appendix Sec.** A and **Sec.** B.

ACKNOWLEDGMENTS

This research was supported by Ant Group (CCF-Ant Research Fund, Ant Group Research Fund).

REFERENCES

[1] Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, et al. 2024. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. *arXiv preprint arXiv:2407.00121* (2024).

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

[3] Fireworks AI. 2024. FireFunction v1. https://huggingface.co/fireworks-ai/firefunction-v1. Function Calling Large Language Model.

[4] LangChain AI. 2024. LangChain: Build context-aware reasoning applications. https://github.com/langchain-ai/langchain.

[5] Mistral AI. 2024. Function Calling Documentation. https://docs.mistral.ai/capabilities/function_calling/. LLM Function Calling Technical Documentation.

[6] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–37.

[7] Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. *arXiv preprint arXiv:1909.00109* (2019).

[8] Anthropic. 2024. *The Claude 3 Model Family: Opus, Sonnet, Haiku.* Model Card. Anthropic. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf

[9] Anthropic. 2024. Tool Use with Claude. https://docs.anthropic.com/en/docs/build-with-claude/tool-use. Documentation on Claude's function calling capabilities.

[10] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[11] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. *arXiv preprint arXiv:1707.02275* (2017).

[12] Kinjal Basu, Ibrahim Abdelaziz, Kelsey Bradford, Maxwell Crouse, Kiran Kate, Sadhana Kumaravel, Saurabh Goyal, Asim Munawar, Yara Rizk, Xin Wang, et al. 2024. NESTFUL: A Benchmark for Evaluating LLMs on Nested Sequences of API Calls. *arXiv preprint arXiv:2409.03797* (2024).

[13] Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A Lastras. 2024. API-BLEND: A Comprehensive Corpora for Training and Benchmarking API LLMs. *arXiv preprint arXiv:2402.15491* (2024).

[14] Nirav Bhan, Shival Gupta, Sai Manaswini, Ritik Baba, Narun Yadav, Hillori Desai, Yash Choudhary, Aman Pawar, Sarthak Shrivastava, and Sudipta Biswas. 2024. Benchmarking Floworks against OpenAI & Anthropic: A Novel Framework for Enhanced LLM Function Calling. *arXiv preprint*

arXiv:2410.17950 (2024).

[15] Matthew Blackwell, Stefano Iacus, Gary King, and Giuseppe Porro. 2009. cem: Coarsened exact matching in Stata. *The Stata Journal* 9, 4 (2009), 524–546.

[16] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* (2020).

[17] Huiying Cao, Yiqun Zhang, Shi Feng, Xiaocui Yang, Daling Wang, and Yifei Zhang. 2024. TOOL-ED: Enhancing Empathetic Response Generation with the Tool Calling Capability of LLM. *arXiv preprint arXiv:2412.03096* (2024).

[18] Shuxiang Cao, Zijian Zhang, Mohammed Alghadeer, Simone D Fasciati, Michele Piscitelli, Mustafa Bakr, Peter Leek, and Alán Aspuru-Guzik. 2024. Agents for self-driving laboratories applied to quantum computing. *arXiv preprint arXiv:2412.07978* (2024).

[19] Robin Chan, Katsiaryna Mirylenka, Thomas Gschwind, Christoph Miksovic, Paolo Scotton, Enrico Toniato, and Abdel Labbi. 2024. Adapting LLMs for Structured Natural Language API Integration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 991–1000.

[20] Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. 2024. Facilitating Multi-turn Function Calling for LLMs via Compositional Instruction Tuning. *arXiv preprint arXiv:2410.12952* (2024).

[21] Wei Chen. 2023. Enhancing User Experience with Extended LLM Functionalities. *User Interface Journal* 19 (2023), 334–347.

[22] Wei Chen and Zhiyuan Li. 2024. Octopus v4: Graph of language models. *arXiv preprint arXiv:2404.19296* (2024).

[23] Yuhan Chen, Ang Lv, Ting-En Lin, Changyu Chen, Yuchuan Wu, Fei Huang, Yongbin Li, and Rui Yan. 2023. Fortify the shortest stave in attention: Enhancing context awareness of large language models for effective tool use. *arXiv preprint arXiv:2312.04455* (2023).

[24] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 9510–9529.

[25] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30 (2017).

[26] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

[27] Cohere. 2024. Command R+ Documentation. https://docs.cohere.com/v2/docs/command-r-plus. Large Language Model Technical Documentation.

[28] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[29] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[30] DeepSeek Inc. 2025. *Function Calling API Guide*. DeepSeek API Documentation. https://api-docs.deepseek.com/guides/function_calling

[31] Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, Rui Yan, et al. 2024. Mobile-bench: An evaluation benchmark for llm-based mobile agents. *arXiv preprint arXiv:2407.00993* (2024).

[32] Yujuan Ding, Wenqi Fan, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meets llms: Towards retrieval-augmented large language models. *arXiv preprint arXiv:2405.06211* (2024).

[33] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 320–335.

[34] Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. 2024. Exploiting LLM Quantization. *arXiv preprint arXiv:2405.18137* (2024).

[35] Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2024. Tinyagent: Function calling at the edge. *arXiv preprint arXiv:2409.00608* (2024).

[36] Hugging Face. 2023. Transformers Agents Documentation. https://huggingface.co/docs/transformers/transformers_agents Accessed: 2024-07-18.

[37] Michael Fore, Simranjit Singh, and Dimitrios Stamoulis. 2024. GeckOpt: LLM System Efficiency via Intent-Based Tool Selection. In *Proceedings of the Great Lakes Symposium on VLSI 2024*. 353–354.

[38] Silin Gao, Jane Dwivedi-Yu, Ping Yu, Xiaoqing Ellen Tan, Ramakanth Pasunuru, Olga Golovneva, Koustuv Sinha, Asli Celikyilmaz, Antoine Bosselut, and Tianlu Wang. 2024. Efficient Tool Use with Chain-of-Abstraction Reasoning. *arXiv preprint arXiv:2401.17464* (2024).

[39] Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 18030–18038.

[40] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[41] Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023. Llm as os (llmao), agents as apps: Envisioning aios, agents and the aios-agent ecosystem. *arXiv preprint arXiv:2312.03815* (2023).

[42] Jian Guan, Wei Wu, Zujie Wen, Peng Xu, Hongning Wang, and Minlie Huang. 2024. AMOR: A Recipe for Building Adaptable Modular Knowledge Agents Through Process Feedback. *arXiv preprint arXiv:2402.01469* (2024).

[43] Anchun Gui, Jian Li, Yong Dai, Nan Du, and Han Xiao. 2024. Look Before You Leap: Towards Decision-Aware and Generalizable Tool-Usage for Large Language Models. *arXiv preprint arXiv:2402.16696* (2024).

[44] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[45] Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Hao Xiong, and Wenliang Chen. 2024. NesTools: A Dataset for Evaluating Nested Tool Learning Abilities of Large Language Models. *arXiv preprint arXiv:2410.11805* (2024).

[46] Senyu Han, Lu Chen, Li-Min Lin, Zhengshan Xu, and Kai Yu. 2024. IBSEN: Director-Actor Agent Collaboration for Controllable and Interactive Drama Script Generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1607–1619.

[47] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems* 36 (2024).

[48] Yupu Hao, Pengfei Cao, Zhuoran Jin, Huanxuan Liao, Yubo Chen, Kang Liu, and Jun Zhao. 2024. CITI: Enhancing Tool Utilizing Ability in Large Language Models without Sacrificing General Performance. *arXiv preprint arXiv:2409.13202* (2024).

[49] Jáchym Herynek and Stefan Edelkamp. 2024. Heuristic Planner for Communication-Constrained Multi-Agent Multi-Goal Path Planning. *arXiv preprint arXiv:2412.13719* (2024).

[50] Tenghao Huang, Dongwon Jung, and Muhao Chen. 2024. Planning and Editing What You Retrieve for Enhanced Tool Learning. *arXiv preprint arXiv:2404.00450* (2024).

[51] Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. 2024. QueryAgent: A Reliable and Efficient Reasoning Framework with Environmental Feedback based Self-Correction. *arXiv preprint arXiv:2403.11886* (2024).

[52] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716* (2024).

[53] Zhongzhen Huang, Kui Xue, Yongqi Fan, Linjie Mu, Ruoyu Liu, Tong Ruan, Shaoting Zhang, and Xiaofan Zhang. 2024. Tool Calling: Enhancing Medication Consultation via Retrieval-Augmented Large Language Models. *arXiv preprint arXiv:2404.17897* (2024).

[54] Wolfram Research, Inc. [n. d.]. Mathematica, Version 14.0. https://www.wolfram.com/mathematica Champaign, IL, 2024.

[55] Alon Jacovi, Avi Caciularu, Jonathan Herzig, Roee Aharoni, Bernd Bohnet, and Mor Geva. 2023. A comprehensive evaluation of tool-assisted generation strategies. *arXiv preprint arXiv:2310.10062* (2023).

[56] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[57] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating LLM hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 1827–1843.

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

[59] Song Jiang, Da JU, Andrew Cohen, Sasha Mitts, Aaron Foss, Justine T Kao, Xian Li, and Yuandong Tian. 2024. Towards Full Delegation: Designing Ideal Agentic Behaviors for Travel Planning. *arXiv preprint arXiv:2411.13904* (2024).

[60] Ulas Berk Karli, Juo-Tung Chen, Victor Nikhil Antony, and Chien-Ming Huang. 2024. Alchemist: LLM-Aided End-User Development of Robot Applications. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*. 361–370.

[61] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An LLM compiler for parallel function calling. *arXiv preprint arXiv:2312.04511* (2023).

[62] Woojeong Kim, Ashish Jagmohan, and Aditya Vempaty. 2024. SEAL: Suite for Evaluating API-use of LLMs. *arXiv preprint arXiv:2409.15523* (2024).

[63] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[64] Yilun Kong, Jingqing Ruan, Yihong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Guoqing Du, Xiaoru Hu, Hangyu Mao, Ziyue Li, et al. 2023. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. *arXiv preprint arXiv:2311.11315* (2023).

[65] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.

[66] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational linguistics* 39, 4 (2013), 885–916.

[67] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 188–197.

[68] Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology* (2023).

[69] Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2023. Tool-augmented reward modeling. *arXiv preprint arXiv:2310.01045* (2023).

[70] Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244* (2023).

[71] Yang Li, Yangyang Yu, Haohang Li, Zhi Chen, and Khaldoun Khashanah. 2023. TradingGPT: Multi-agent system with layered memory and distinct characters for enhanced financial trading performance. *arXiv preprint arXiv:2309.03736* (2023).

[72] Yuan Li, Yixuan Zhang, and Lichao Sun. 2023. Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents. *arXiv preprint arXiv:2310.06500* (2023).

[73] Zeyang Li and Navid Azizan. 2024. Safe Multi-Agent Reinforcement Learning with Convergence to Generalized Nash Equilibrium. *arXiv preprint arXiv:2411.15036* (2024).

[74] Xinnian Liang, Bing Wang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. 2023. Unleashing infinite-length input capacity for large-scale language models with self-controlled memory system. *arXiv e-prints* (2023), arXiv–2304.

[75] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing* 3 (2024), 0063.

[76] John Licato, Logan Fields, Stephen Steinle, and Brayden Hollis. 2024. Shot Selection to Determine Legality of Actions in a Rule-Heavy Environment. In *2024 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[77] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[78] Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. 2024. Hammer: Robust Function-Calling for On-Device Language Models via Function Masking. *arXiv preprint arXiv:2410.04587* (2024).

[79] Jiacheng Liu, Skyler Hallinan, Ximing Lu, Pengfei He, Sean Welleck, Hannaneh Hajishirzi, and Yejin Choi. 2022. Rainier: Reinforced knowledge introspector for commonsense question answering. *arXiv preprint arXiv:2210.03078* (2022).

[80] Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. 2023. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint arXiv:2311.08719* (2023).

[81] Ruicheng Liu, Rui Mao, Anh Tuan Luu, and Erik Cambria. 2023. A brief survey on recent advances in coreference resolution. *Artificial Intelligence Review* 56, 12 (2023), 14439–14481.

[82] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920* (2024).

[83] Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and XiangAnthony' Chen. 2024. Proactive Conversational Agents with Inner Thoughts. *arXiv preprint arXiv:2501.00383* (2024).

[84] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. 2024. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518* (2024).

[85] Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Zhiheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. 2023. Controlllm: Augment language models with tools by searching on graphs. *arXiv preprint arXiv:2310.17796* (2023).

[86] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *arXiv preprint arXiv:2401.13178* (2024).

[87] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems* 36 (2023), 21702–21720.

[88] Zixian Ma, Jianguo Zhang, Zhiwei Liu, Jieyu Zhang, Juntao Tan, Manli Shu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Caiming Xiong, et al. 2024. TACO: Learning Multi-modal Action Models with Synthetic Chains-of-Thought-and-Action. *arXiv preprint arXiv:2412.05479* (2024).

[89] Graziano A Manduzio, Federico A Galatolo, Mario GCA Cimino, Enzo Pasquale Scilingo, and Lorenzo Cominelli. 2024. Improving Small-Scale Large Language Models Function Calling for Reasoning Tasks. *arXiv preprint arXiv:2410.18890* (2024).

[90] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842* (2023).

[91] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 932–949.

[92] Microsoft. 2024. Semantic Kernel. https://github.com/microsoft/semantic-kernel.

[93] Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, et al. 2023. Orca 2: Teaching small language models how to reason. *arXiv preprint arXiv:2311.11045* (2023).

[94] Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2023. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322* (2023).

[95] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707* (2023).

[96] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Bhuminand Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Compact language models via pruning and knowledge distillation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[97] Yohei Nakajima. 2024. BabyAGI: An experimental framework for self-building autonomous agents. https://github.com/yoheinakajima/babyagi.

[98] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).

[99] Deepak Nathani, David Wang, Liangming Pan, and William Yang Wang. 2023. MAF: Multi-aspect feedback for improving reasoning in large language models. *arXiv preprint arXiv:2310.12426* (2023).

[100] Xuan-Phi Nguyen, Shrey Pandit, Senthil Purushwalkam, Austin Xu, Hailin Chen, Yifei Ming, Zixuan Ke, Silvio Savarese, Caiming Xong, and Shafiq Joty. 2024. Sfr-rag: Towards contextually faithful llms. *arXiv preprint arXiv:2409.09916* (2024).

[101] Kaiwen Ning, Jiachi Chen, Jingwen Zhang, Wei Lia, Zexu Wang, Yuming Feng, Weizhe Zhang, and Zibin Zheng. 2024. Defining and Detecting the Defects of the Large Language Model-based Autonomous Agents. *arXiv preprint arXiv:2412.18371* (2024).

[102] Kangyun Ning, Yisong Su, Xueqiang Lv, Yuanzhe Zhang, Jian Liu, Kang Liu, and Jinan Xu. 2024. Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models. *arXiv preprint arXiv:2407.12823* (2024).

[103] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114* (2021).

[104] OpenAI. 2023. ChatGPT Plugins. https://openai.com/blog/chatgpt-plugins Accessed: 2024-07-18.

[105] OpenAI. 2023. Introducing Function Calling in ChatGPT. https://openai.com/blog/function-calling. Accessed: 2023-10-01.

[106] Yasser Otiefy and Alaa Alhamzeh. 2024. Exploring Large Language Models in Financial Argument Relation Identification. In *Proceedings of the Joint Workshop of the 7th Financial Technology and Natural Language Processing, the 5th Knowledge Discovery from Unstructured Data in Financial Services, and the 4th Workshop on Economics and Natural Language Processing@ LREC-COLING 2024*. 119–129.

[107] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[108] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560* (2023).

[109] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[110] Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255* (2022).

[111] Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. 2024. Grammar-Aligned Decoding. *arXiv preprint arXiv:2405.21047* (2024).

[112] Shishir Patil et al. 2024. Berkeley Function Calling Leaderboard (BFCL). https://github.com/ShishirPatil/gorilla/tree/main/berkeley-function-call-leaderboard.

[113] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334* (2023).

[114] Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Cheng Yang, Xin Cong, Xiaoyin Che, et al. 2023. Experiential co-learning of software-developing agents. *arXiv preprint arXiv:2312.17025* (2023).

[115] Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, et al. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. *arXiv preprint arXiv:2402.09205* (2024).

[116] Cheng Qian, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. 2023. Toolink: Linking toolkit creation and using through chain-of-solving on open-source model. *arXiv preprint arXiv:2310.05155* (2023).

[117] Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2022. Reasoning with language model prompting: A survey. *arXiv preprint arXiv:2212.09597* (2022).

[118] Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. 2023. Webcpm: Interactive web search for chinese long-form question answering. *arXiv preprint arXiv:2305.06849* (2023).

[119] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, et al. 2023. BMTools: Tool Learning for Big Models, Open-Source Solutions of ChatGPT-Plugins. https://github.com/OpenBMB/BMTools. Apache-2.0 license.

[120] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[121] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. ShowUI: One Vision-Language-Action Model for GUI Visual Agent. *arXiv e-prints* (2024), arXiv–2411.

[122] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. COLT: Towards Completeness-Oriented Tool Retrieval for Large Language Models. *arXiv preprint arXiv:2405.16089* (2024).

[123] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool Learning with Large Language Models: A Survey. *arXiv preprint arXiv:2405.17935* (2024).

[124] Qwen Team. 2024. *Function Calling - Qwen*. Qwen Documentation. https://qwen.readthedocs.io/en/latest/framework/function_call.html

[125] Qwen Team. 2025. *QwQ-32B: Embracing the Power of Reinforcement Learning*. Qwen. https://qwenlm.github.io/blog/qwq-32b/

[126] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446* (2021).

[127] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2024).

[128] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[129] Gentiana Rashiti, Geethan Karunaratne, Mrinmaya Sachan, Abu Sebastian, and Abbas Rahimi. 2024. Retro-li: Small-Scale Retrieval Augmented Generation Supporting Noisy Similarity Searches and Domain Shift Generalization. *arXiv preprint arXiv:2410.00004* (2024).

[130] Nous Research, Teknium, and Karan4D. 2024. Nous-Hermes-13b: A Fine-tuned Language Model. https://huggingface.co/NousResearch/Nous-Hermes-13b. LLaMA-13B Based Instruction-tuned Model.

[131] Salesforce Research. 2024. XLAM Function Calling Dataset. https://huggingface.co/datasets/Salesforce/xlam-function-calling-60k. 60K function calling training dataset.

[132] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[133] Kaushik Roy, Yuxin Zi, Chathurangi Shyalika, Renjith Prasad, Sidhaarth Murali, Vedant Palit, and Amit Sheth. 2024. QA-RAG: Leveraging Question and Answer-based Retrieved Chunk Re-Formatting for Improving Response Quality During Retrieval-augmented Generation. (2024).

[134] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427* (2023).

[135] Avirup Saha, Lakshmi Mandal, Balaji Ganesan, Sambit Ghosh, Renuka Sindhgatta, Carlos Eberhardt, Dan Debrunner, and Sameep Mehta. 2024. Sequential API Function Calling Using GraphQL Schema. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 19452–19458.

[136] Sagar Srinivas Sakhinana, Geethan Sannidhi, and Venkataramana Runkana. 2024. Retrieval-Augmented Instruction Tuning for Automated Process Engineering Calculations: A Tool-Chaining Problem-Solving Framework with Attributable Reflection. *arXiv preprint arXiv:2408.15866* (2024).

[137] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).

[138] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761* (2023).

[139] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227* (2025).

[140] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* (2017).

[141] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608* (2024).

[142] Jie-Jing Shao, Xiao-Wen Yang, Bo-Wen Zhang, Baizhi Chen, Wei-Da Wei, Lan-Zhe Guo, and Yu-feng Li. 2024. ChinaTravel: A Real-World Benchmark for Language Agents in Chinese Travel Planning. *arXiv preprint arXiv:2412.13682* (2024).

[143] ChengAo Shen, Zhengzhang Chen, Dongsheng Luo, Dongkuan Xu, Haifeng Chen, and Jingchao Ni. 2024. Exploring Multi-Modal Integration with Tool-Augmented LLM Agents for Precise Causal Discovery. *arXiv preprint arXiv:2412.13667* (2024).

[144] Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. 2024. Shortcutsbench: A large-scale real-world benchmark for api-based agents. *arXiv preprint arXiv:2407.00132* (2024).

[145] Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024. Small llms are weak tool learners: A multi-llm agent. *arXiv preprint arXiv:2401.07324* (2024).

[146] Yuchen Shi, Siqi Cai, Zihan Xu, Yuei Qin, Gang Li, Hang Shao, Jiawei Chen, Deqing Yang, Ke Li, and Xing Sun. 2025. FlowAgent: Achieving Compliance and Flexibility for Workflow Agents. *arXiv preprint arXiv:2502.14345* (2025).

[147] Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. *arXiv preprint arXiv:2403.03031* (2024).

[148] Significant-Gravitas. 2024. AutoGPT: Accessible AI for everyone. https://github.com/Significant-Gravitas/AutoGPT.

[149] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11523–11530.

[150] Simranjit Singh, Michael Fore, Andreas Karatzas, Chaehong Lee, Yanan Jian, Longfei Shangguan, Fuxun Yu, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. 2024. LLM-dCache: Improving Tool-Augmented LLMs with GPT-Driven Localized Data Caching. *arXiv preprint arXiv:2406.06799* (2024).

[151] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2998–3009.

[152] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. *arXiv preprint arXiv:2306.06624* (2023).

[153] Sakhinana Sagar Srinivas, Vijay Sri Vaikunth, and Venkataramana Runkana. 2024. Knowledge Graph Modeling-Driven Large Language Model Operating System (LLM OS) for Task Automation in Process Engineering Problem-Solving. *arXiv preprint arXiv:2408.14494* (2024).

[154] Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

[155] Gita Sukthankar, Christopher Geib, Hung Hai Bui, David Pynadath, and Robert P Goldman. 2014. *Plan, activity, and intent recognition: Theory and practice*. Newnes.

[156] Ryan Sullivan, Ryan Pégoud, Ameen Ur Rahmen, Xinchen Yang, Junyun Huang, Aayush Verma, Nistha Mitra, and John P Dickerson. 2024. Syllabus: Portable Curricula for Reinforcement Learning Agents. *arXiv preprint arXiv:2411.11318* (2024).

[157] Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, et al. 2023. A survey of reasoning with foundation models. *arXiv preprint arXiv:2312.11562* (2023).

[158] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).

[159] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. 2024. OS-Genesis: Automating GUI Agent Trajectory Construction via Reverse Task Synthesis. *arXiv preprint arXiv:2412.19723* (2024).

[160] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. *arXiv preprint arXiv:2306.05301* (2023).

[161] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[162] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085* (2022).

[163] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

[164] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[165] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901* (2024).

[166] Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2024. Improving llm code generation with grammar augmentation. *arXiv preprint arXiv:2403.01632* (2024).

[167] Mart van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319* (2024).

[168] Ashish Vaswani et al. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[169] vLLM Contributors. 2024. Easy, fast, and cheap LLM serving for everyone |. https://github.com/vllm-project/vllm.

[170] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[171] Pei Wang, Yanan Wu, Zekun Wang, Jiaheng Liu, Xiaoshuai Song, Zhongyuan Peng, Ken Deng, Chenchen Zhang, Jiakai Wang, Junran Peng, et al. 2024. MTU-Bench: A Multi-granularity Tool-Use Benchmark for Large Language Models. *arXiv preprint arXiv:2410.11710* (2024).

[172] Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2024. ToolGen: Unified Tool Retrieval and Calling via Generation. *arXiv preprint arXiv:2410.03439* (2024).

[173] Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, and Michael R Lyu. 2024. Learning to Ask: When LLMs Meet Unclear Instruction. *arXiv preprint arXiv:2409.00557* (2024).

[174] Xin Wang, Yuwei Zhou, Hong Chen, and Wenwu Zhu. 2024. Curriculum Learning: Theories, Approaches, Applications, Tools, and Future Directions in the Era of Large Language Models. In *Companion Proceedings of the ACM on Web Conference 2024*. 1306–1310.

[175] Yaru Wang, Choosuk Jung, Yushu Pan, Jiaao He, Dongsheng Wang, Di Wu, Yuqing Yang, Yanxia Zhang, Yihe Deng, Xinyue Shen, Xiang Yue, Nianli Chen, Song Wei, Dian Yu, Kai Chen, Qipeng Guo, Zheng Zhang, and Qi Zhang. 2024. LlamaFactory: Unified Efficient Fine-tuning of 100+ Language Models. arXiv:2402.00744 [cs.CL]

[176] Yuanchun Wang, Jifan Yu, Zijun Yao, Jing Zhang, Yuyang Xie, Shangqing Tu, Yiyang Fu, Youhe Feng, Jinkai Zhang, Jingyao Zhang, et al. 2024. A Solution-based LLM API-using Methodology for Academic Information Seeking. *arXiv preprint arXiv:2405.15165* (2024).

[177] Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. 2024. Ponder & Press: Advancing Visual GUI Agent towards General Computer Control. *arXiv preprint arXiv:2412.01268* (2024).

[178] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. 2024. Describe, explain, plan and select: interactive planning with LLMs enables open-world multi-task agents. *Advances in Neural Information Processing Systems* 36 (2024).

[179] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452* (2024).

[180] Zekun Wang, Ge Zhang, Kexin Yang, Ning Shi, Wangchunshu Zhou, Shaochun Hao, Guangzheng Xiong, Yizhi Li, Mong Yuan Sim, Xiuying Chen, et al. 2023. Interactive natural language processing. *arXiv preprint arXiv:2305.13246* (2023).

[181] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[182] Feng Wu, Ning Zhang, Somesh Jha, Patrick McDaniel, and Chaowei Xiao. 2024. A new era in LLM security: Exploring security concerns in real-world LLM-based systems. *arXiv preprint arXiv:2402.18649* (2024).

[183] Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-Tools: Self-Instruct Tool Learning Dataset for Agent Tuning and Detailed Benchmark. *arXiv preprint arXiv:2405.08355* (2024).

[184] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024. AvaTaR: Optimizing LLM Agents for Tool-Assisted Knowledge Retrieval. *arXiv preprint arXiv:2406.11200* (2024).

[185] Yuwei Wu, Xuezhe Ma, and Diyi Yang. 2021. Personalized response generation via generative split memory network. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1956–1970.

[186] Zihui Wu, Haichang Gao, Jianping He, and Ping Wang. 2024. The dark side of function calling: Pathways to jailbreaking large language models. *arXiv preprint arXiv:2407.17915* (2024).

[187] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. 2024. OS-ATLAS: A Foundation Action Model for Generalist GUI Agents. *arXiv preprint arXiv:2410.23218* (2024).

[188] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[189] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. [n. d.]. TravelPlanner: A Benchmark for Real-World Planning with Language Agents. In *Forty-first International Conference on Machine Learning*.

[190] Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408* (2023).

[191] Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. 2024. TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks. *arXiv preprint arXiv:2412.14161* (2024).

[192] Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. *arXiv preprint arXiv:2406.17465* (2024).

[193] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116* (2024).

[194] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2024. AndroidLab: Training and Systematic Benchmarking of Android Autonomous Agents. *arXiv preprint arXiv:2410.24024* (2024).

[195] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671* (2024).

[196] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115* (2024).

[197] Chuanpeng Yang, Yao Zhu, Wang Lu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, and Yiqiang Chen. 2024. Survey on knowledge distillation for large language models: Methods, evaluation, and application. *ACM Transactions on Intelligent Systems and Technology* (2024).

[198] Lingfeng Yang, Zhenyuan Chen, Xiang Li, Peiyang Jia, Liangqu Long, and Jian Yang. 2024. Agent-based Video Trimming. *arXiv preprint arXiv:2412.09513* (2024).

[199] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems* 36 (2024).

[200] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. 2023. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129* (2023).

[201] Zongxin Yang, Guikun Chen, Xiaodi Li, Wenguan Wang, and Yi Yang. 2024. Doraemongpt: Toward understanding dynamic scenes with large language models (exemplified as a video agent). In *Forty-first International Conference on Machine Learning*.

[202] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771* (2023).

[203] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[204] Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. 2024. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741* (2024).

[205] Junjie Ye, Yilong Wu, Songyang Gao, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. RoTBench: A Multi-Level Benchmark for Evaluating the Robustness of Large Language Models in Tool Learning. *arXiv preprint arXiv:2401.08326* (2024).

[206] Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

[207] Botao Yu, Frazier N Baker, Ziru Chen, Garrett Herb, Boyu Gou, Daniel Adu-Ampratwum, Xia Ning, and Huan Sun. 2024. Tooling or Not Tooling? The Impact of Tools on Language Agents for Chemistry Problem Solving. *arXiv preprint arXiv:2411.07228* (2024).

[208] Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. 2023. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428* (2023).

[209] Ye Yuan, Chengwu Liu, Jingyang Yuan, Gongbo Sun, Siqi Li, and Ming Zhang. 2024. A Hybrid RAG System with Comprehensive Enhancement on Complex Reasoning. *arXiv preprint arXiv:2408.05141* (2024).

[210] Yankai Lin Weize Chen Ning Ding Ganqu Cui Zheni Zeng Yufei Huang Chaojun Xiao Chi Han Yi Ren Fung Yusheng Su Huadong Wang Cheng Qian Runchu Tian Kunlun Zhu Shihao Liang Xingyu Shen Bokai Xu Zhen Zhang Yining Ye Bowen Li Ziwei Tang Jing Yi Yuzhang Zhu Zhenning Dai Lan Yan Xin Cong Yaxi Lu Weilin Zhao Yuxiang Huang Junxi Yan Xu Han Xian Sun Dahai Li Jason Phang Cheng Yang Tongshuang Wu Heng Ji Zhiyuan Liu Maosong Sun Yujia Qin, Shengding Hu. 2023. Tool Learning with Foundation Models. *arXiv preprint arXiv:2304.08354* (2023).

[211] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823* (2023).

[212] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: An Open Bilingual Pre-trained Model. In *The Eleventh International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=-Aw0rrrPUF

[213] Bo Zhang, Yi Tan, Yu Shen, et al. 2024. Breaking Agents: Compromising Autonomous LLM Agents Through Malfunction Amplification. *arXiv preprint arXiv:2407.20859* (2024).

[214] Hongming Zhang, Yan Song, Yangqiu Song, and Dong Yu. 2019. Knowledge-aware Pronoun Coreference Resolution. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 867.

[215] Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. 2024. AgentOhana: Design Unified Data and Training Pipeline for Effective Agent Learning. *arXiv preprint arXiv:2402.15506* (2024).

[216] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? *arXiv preprint arXiv:1801.07243* (2018).

[217] Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. Agent-pro: Learning to evolve via policy-level reflection and optimization. *arXiv preprint arXiv:2402.17574* (2024).

[218] Yinger Zhang, Hui Cai, Xeirui Song, Yicheng Chen, Rui Sun, and Jing Zheng. 2023. Reverse chain: A generic-rule for llms to master multi-api planning. *arXiv preprint arXiv:2310.04474* (2023).

[219] Yipeng Zhang, Xin Wang, Hong Chen, Jiapei Fan, Weigao Wen, Hui Xue, Hong Mei, and Wenwu Zhu. 2024. Large Language Model with Curriculum Reasoning for Visual Concept Recognition. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6269–6280.

[220] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).

[221] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[222] Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems* 36 (2024).

[223] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

[224] Qiaoyu Zheng, Chaoyi Wu, Pengcheng Qiu, Lisong Dai, Ya Zhang, Yanfeng Wang, and Weidi Xie. 2024. Can Modern LLMs Act as Agent Cores in Radiology˜ Environments? *arXiv preprint arXiv:2412.09529* (2024).

[225] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19724–19731.

[226] Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2024. Efficient Multi-Agent Collaboration with Tool Use for Online Planning in Complex Table Question Answering. *arXiv preprint arXiv:2412.20145* (2024).

[227] Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. 2024. Proposer-Agent-Evaluator (PAE): Autonomous Skill Discovery For Foundation Model Internet Agents. *arXiv preprint arXiv:2412.13194* (2024).

[228] Mu Zhu. 2004. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2, 30 (2004), 6.

[229] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. Toolchain*: Efficient action space navigation in large language models with a* search. *arXiv preprint arXiv:2310.13227* (2023).

[230] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).

[231] Rui Zuo, Zifan Wang, Simon Khan, Garrett Ethan Katz, and Qinru Qiu. 2024. Why the Agent Made that Decision: Explaining Deep Reinforcement Learning with Vision Masks. *arXiv preprint arXiv:2411.16120* (2024).

A ASSESSING THE ABILITY: EVALUATION OF FUNCTION CALLING TASKS

Evaluating the capabilities of function calling in large language models (LLMs) is a multifaceted challenge due to the numerous steps involved in the process. It is insufficient to judge a function-calling system from a single perspective, and currently, there is no comprehensive evaluation metric that fully captures its performance. Existing evaluation methods either focus on specific components or calculate aggregated scores across various tasks using large benchmarks. As shown in Table 4, we summarize both the commonly used evaluation metrics and major benchmarks that have been developed to assess function calling capabilities in LLMs.

Table 4. Overview of Evaluation Methods and Benchmarks for Function Calling in LLMs

Category	Evaluation Type	Methods and Benchmarks
Overall Performance (§A.1)	Function Selection Metrics	Recall@K [228], NDCG@K [56], COMP@K
	Core Evaluation Metrics	Pass Rate [120], Win/Success Rate (information richness, factual accuracy, reasoning quality) [12]
	Comprehensive Assessment	T-Eval (planning, reasoning, retrieval, understanding, instruction following, review) [24]
	Quality-based Metrics	BLEU [109], ROUGE-L [77], Exact Match [15], F1 score [13]
Benchmarks (§A.2)	Early Foundational	ToolLLM [120], ToolAlpaca [160], Gorilla [113]
	Standardized Platforms	APIBench [113], API-Bank [70]
	Domain-Specific	ShortcutsBench [144], BigCodeBench [230], SEAL [62], RadABench [224], NoisyToolBench [173], Mobile-Bench [31]
	Task-Oriented	IN3 [115], NESTFUL [12], UltraTool [50], AppWorld [165], TheAgentCompany [191], AgentBoard [86], TravelPlanner [189], ChinaTravel [142]
	Comprehensive Systems	API-BLEND [13], NESTOOLS [45], MTU-Bench [171], WTU-EVAL [102]

A.1 Overall Performance

A.1.1 ***Function Selection Metrics.*** Several metrics are used to evaluate function selection effectiveness, including Recall@K [228], which measures the proportion of correctly selected functions, NDCG@K [56] which considers both relevance and ranking position, and COMP@K, which assesses whether all necessary functions are included in the top-K selections, where K represents the number of selected functions.

A.1.2 ***Core Evaluation Metrics.*** The **Pass Rate** calculates the proportion of successfully completed instructions within limited budgets [120]. This metric assesses the executability of instructions for a language model in the context of tool usage. However, it is considered a loose metric since the solution path is deemed a pass even if the final answer does not resolve the original instruction after the model has attempted all the available APIs.

The **Win/Success Rate**, on the other hand, evaluates how well an instruction is completed [12]. The criteria for the Win/Success Rate include factors such as information richness, factual accuracy, reasoning quality, milestones reached during execution, exploration of potentially useful APIs, and minimization of repeated or redundant API calls. This metric provides a more nuanced assessment of the model's performance by considering the quality and efficiency of the solutions generated. Pass Rate and Win/Success Rate serve as fundamental metrics in evaluating LLM performance, with Pass Rate focusing on task completion and Win/Success Rate assessing solution quality. These complementary metrics provide a basic framework for understanding model effectiveness in function calling tasks.

A.1.3 ***Comprehensive Assessment Framework.*** T-Eval decomposes the function calling evaluation into six fundamental abilities (planning, reasoning, retrieval, understanding, instruction following, and review) and provides step-by-step assessment protocols to comprehensively evaluate LLMs' tool utilization capabilities [24].

A.1.4 ***Quality-based Evaluation***.

The fundamental objective of tool-learning methodological implementations centers on enhancing operational capabilities of large-scale linguistic computational frameworks to address downstream task requirements with elevated efficacy parameters. Consequently, the operational effectiveness of tool utilization mechanisms undergoes evaluation predominantly through performance metrics in resolving these downstream computational tasks [160]. This evaluative paradigm necessitates that linguistic computational frameworks effectively consolidate informational components acquired throughout the comprehensive operational process, ultimately generating precise and contextually appropriate responses to user interrogative inputs. The qualitative assessment of terminal responses can be quantified through implementation of sophisticated metrics including BLEU scoring methodology [109], ROUGE-L analytical frameworks [77], Exact Match evaluation paradigms [15], F1 statistical measurement protocols [13], among additional evaluative frameworks. These quantitative assessment methodologies evaluate multidimensional aspects including correctness coefficients, completeness parameters, and linguistic qualitative dimensions of computational responses generated by large-scale linguistic models.

A.2 Benchmarks

A.2.1 ***Early Foundational Frameworks***.

The evolution of function-calling benchmarks has seen substantial advancements through several pioneering evaluation frameworks. The ToolLLM methodology [120] established a systematic approach for assessing tool-augmented language models' capabilities, providing a systematic approach to evaluating tool-use capabilities. This foundation was enhanced by ToolAlpaca's specialized metrics [160], which focused on analyzing the alignment between model outputs and tool specifications in instruction-following tasks, specifically analyzing alignment between model outputs and tool specifications. These foundational works have significantly influenced current evaluations of LLMs' function-calling capabilities.

A.2.2 ***Standardized Evaluation Platforms***.

The field has witnessed the emergence of comprehensive evaluation platforms, notably through the Berkeley Function Calling Leaderboard developed via Gorilla [113]. This computational integration platform facilitates authentic application programming interface operational interactions while maintaining systematic updating protocols, whereas APIBench [113] has substantially enhanced its precision coefficients through implementation of sophisticated analytical methodologies including Abstract Syntax Tree (AST) structural evaluation paradigms. The API-Bank architectural framework [70] further expanded evaluative capabilities through implementation of robust validation infrastructure for programmatic interface interaction mechanisms. Building upon foundational benchmarking architectures such as ToolLLM [120] and ToolAlpaca [160], API-Bank encompasses a more comprehensive functional spectrum, incorporating enterprise-grade and open-source technological implementations, while simultaneously addressing multifaceted complexities inherent within multi-iterative conversational exchanges. Its comprehensive methodological approach quantifies multiple operational dimensions ranging from parameter extraction protocols to sophisticated dialogical management capabilities, providing exhaustive analytical insights regarding linguistic model functional invocation performance metrics. Contemporary benchmarking architectural frameworks, exemplified through Gorilla [113] and API-Bank [70], emphasize authentic operational relevance through multidimensional evaluative methodologies. These assessment protocols examine directive comprehension capabilities, tool selection accuracy coefficients, parameter configuration precision metrics, and execution validation methodologies. By incorporating enterprise functions and open-source tools, as pioneered by ToolLLM [120], these frameworks simulate realistic deployment conditions.

A.2.3 ***Domain-Specific Benchmarks.*** The field has developed specialized benchmarks for specific application domains. ShortcutsBench [144] has emerged as a large-scale benchmark specifically designed for evaluating API-based agents, featuring real APIs from Apple's operating systems and human-annotated action sequences, with a focus on tasks of varying difficulty levels and parameter-filling capabilities. BigCodeBench [230] presents a rigorous evaluation framework for assessing LLMs' function-calling capabilities through 1,140 practical programming tasks across 139 libraries and 7 domains, introducing both structured docstring-based completion and natural language instruction-based generation scenarios to comprehensively measure models' ability to understand and implement complex function calls. SEAL [62] proposes a comprehensive tool evaluation framework that standardizes benchmarks, introduces a GPT-4-powered API simulator with caching, and provides end-to-end evaluation of API retrieval, calls and responses. RadABench [224] evaluates whether current LLMs can serve as agent cores in radiology by introducing RadABench-Data with diverse cases across 6 anatomies/5 modalities/11 tasks, and RadABench-EvalPlat featuring dynamic tool set simulation. NoisyToolBench [173] serves as a challenging benchmark for evaluating LLMs' tool use under imperfect instructions by systematically covering four types of instruction issues: missing key information, multiple references, errors, and beyond tool capabilities. Mobile-Bench [31] proposes a benchmark for evaluating LLM-based mobile agents in handling interleaved vision-language action streams through both UI operations and API calls.

A.2.4 ***Task-Oriented Evaluation Frameworks.*** Recent frameworks have emerged focusing on specific functional aspects. The IN3 framework [115] proposes a novel benchmark for evaluating and improving intention understanding in function mapping, with comprehensive metrics and annotations that enable effective training and assessment of model-user interaction capabilities. NESTFUL [12] proposes a benchmark focusing on evaluating LLMs' capability to plan and execute nested sequences of API calls, where outputs from earlier calls serve as inputs to subsequent ones, demonstrating the challenges in direct planning for complex API interactions. UltraTool [50] proposes a comprehensive benchmark for evaluating LLMs' capabilities in planning, creating and using tools within real-world complex scenarios, focusing on task decomposition and direct planning without relying on memory augmentation. AppWorld [165] proposes a comprehensive benchmark for interactive code generation with complex API calls, featuring real-world apps and programmatic evaluation of LLMs. TheAgentCompany [191] presents a comprehensive benchmark that evaluates LLM agents on realistic tasks in a software company setting, featuring self-hosted environment with internal websites and structured professional tasks across software engineering, project management, HR and finance domains. The benchmark adopts checkpoint-based evaluation for long-horizon tasks and enables agent-human interaction through simulated colleagues. AgentBoard [86] proposes a comprehensive benchmark and visualization framework for evaluating multi-turn LLM agents through interleaved vision, language and action interactions. TravelPlanner [189] proposes a benchmark for evaluating language agents' planning capability in real-world scenarios, featuring tools for accessing millions of records, explicit and implicit constraints, and comprehensive evaluation metrics for assessing memory-augmented planning abilities. Another notable travel benchmark is ChinaTravel [142], which evaluates language agents on complex multi-day travel planning with authentic Chinese data, featuring a domain-specific language for scalable constraint validation and both synthetic and human-derived queries.

A.2.5 ***Comprehensive Evaluation Systems.*** Recent developments have produced more holistic evaluation approaches. API-BLEND [13] presents a large-scale training and evaluation framework for API-focused LLMs, featuring a hybrid approach to data generation, multi-domain API coverage, and systematic evaluation methods that demonstrate superior out-of-domain generalization compared to existing tool-augmented solutions. NESTOOLS [45] proposes an evaluation benchmark specifically for nested function calling capabilities of LLMs. It uses an automated method to

Table 5. Overview of Industry Products for Function Calling

Industry Products	Commercial Platforms (§ B.1)	ChatGPT plugins [104], Claude's tool use API [9], Cohere Command [27], Qwen [124, 125, 195, 196], and DeepSeek [30, 44]
	Frameworks & SDKs (§ B.2)	HuggingFace Transformer Agents [36], Semantic Kernel [92], LangChain [4], WebCPM [118]
	Autonomous Agent Systems (§ B.3)	Auto-GPT [148], BabyAGI [97], BMTools [119], RestGPT [152], xLAM [131], Octopus-v4 [22]
	Open Source Models (§ B.4)	GRANITE-20B [1], Mistral 7B [5], NexusRaven V2-13B [154], Gorilla [113], FireFunction V1 [3], Nous Hermes 2 [130]
	Training Resources & Datasets (§ B.5)	AgentInstruct [211], AgentOhana [215], Lumos [206]

generate large-scale examples with different nesting structures, ensures quality through manual review and refinement, and provides four evaluation dimensions (tool selection, calling order, parameter filling, and nested parameter filling) to comprehensively assess LLMs' nested tool calling abilities. Experiments on 22 LLMs show that current models still face challenges in handling complex nested tool calls. MTU-Bench [171] introduces a comprehensive multi-granularity tool-use benchmark comprising MTU-Instruct for training and MTU-Eval for automatic evaluation, enabling LLMs to effectively handle diverse tool-use scenarios across single/multi-turn dialogues and single/multi-tool interactions through high-quality instruction data and fine-grained evaluation metrics. WTU-EVAL [102] presents the first evaluation framework to assess whether LLMs can accurately determine the necessity of tool usage, featuring eleven datasets across different domains and demonstrating that most current LLMs struggle with tool usage decision-making, especially in general tasks where tools are not required.

A.2.6 ***Evolution of Performance Metrics.*** Performance metrics have evolved considerably since these benchmarks' inception. The evaluation methodology established by ToolAlpaca [160] and refined through subsequent research [70, 113] now encompasses response time, resource efficiency, and scalability. Cross-platform compatibility testing, an area initially addressed by ToolLLM [120], has become essential for evaluating LLMs' ability to handle various API protocols, authentication mechanisms, and data formats across different services. These technological progressions, with substantial methodological contributions emanating from ToolLLM [120], ToolAlpaca [160], API-Bank [70], and Gorilla [113], have established a robust epistemological infrastructure for the systematic evaluation and enhancement of functional invocation capabilities within contemporary linguistic computational frameworks.

This progress is vital for advancing LLM performance in real-world applications and maintaining rigorous, evolving evaluation standards. AgentBoard [86] introduces novel evaluation frameworks for assessing function-calling capabilities, with AgentBoard focusing on multi-turn tool interactions across nine distinct tasks.

B INDUSTRY PRODUCTS

Current mature industry products capable of function calling include several notable tools, as summarized in Table 5.

B.1 Commercial Platforms

ChatGPT ChatGPT plugins [104] extend the capabilities of the ChatGPT model by enabling it to access and interact with various external applications and services. This allows users to retrieve real-time information, perform tasks, and integrate seamlessly with other tools. Claude's tool use API [9] extends the model's capabilities by enabling controlled interactions with external functions and services. This allows developers to integrate external data sources, execute specific tasks, and enhance Claude's functionality through structured tool interactions. Similarly, Qwen [124, 195, 196], QwQ-32B [125], and DeepSeek [30, 44] have also implemented function calling capabilities, though DeepSeek's current implementation is noted to be unstable with potential issues like loop calls and empty responses.

B.2 Development Frameworks and SDKs

HuggingFace Transformer Agents [36] are particularly powerful for multimodal tasks, enabling large language models (LLMs) to perform a wide range of tasks by leveraging pre-trained models and integrating them with various data modalities such as text, images, and audio. Semantic Kernel [92] is an SDK that integrates LLMs like OpenAI, Azure OpenAI, and Hugging Face with conventional programming languages such as C#, Python, and Java. This facilitates the creation of sophisticated AI applications by allowing developers to seamlessly combine the strengths of LLMs with traditional software development practices. Additionally, LangChain [4] is a versatile framework for developing applications powered by large language models. It facilitates the architectural implementation of sophisticated operational workflows through enabling linguistic computational frameworks to engage in programmatic interface interactions with Application Programming Interfaces, relational and non-relational data repositories, and heterogeneous computational systems, rendering it particularly advantageous for the development of intelligent assistive frameworks and automated procedural systems.

WebCPM [118] constitutes a comprehensive architectural framework engineered specifically for the development of Chinese-language extended-form interrogative-responsive applications through implementation of interactive web-based information retrieval methodologies. This framework enables large-scale linguistic computational models to simulate anthropomorphic web navigation behavioral patterns to extract and synthesize informational content from diversified digital repositories. WebCPM demonstrates particular efficacy in the construction of sophisticated operational workflows wherein linguistic models interact with search engine infrastructures, extract contextually relevant data components, and generate comprehensive informational responses.

B.3 Autonomous Agent Systems

Within contemporary technological progressions, numerous open-source computational libraries and developmental toolkits have been proposed to augment tool-learning methodological implementations within large linguistic models. Auto-GPT [148] constitutes an open-source computational application specifically engineered to facilitate autonomous execution of complex operational tasks by large-scale linguistic models with minimal anthropogenic input requirements. This framework enables computational models to decompose overarching objectives into sequential subtasks and systematically execute these components, incorporating capabilities for accessing distributed information networks and engaging with Application Programming Interfaces.

Correspondingly, BabyAGI [97] represents an open-source architectural framework designed for the development of autonomous artificial intelligence operational agents capable of managing and executing complex task sequences with minimal supervisory requirements. Its modularized architectural design facilitates developer extensibility through integration of supplementary tools or programmatic interfaces, rendering it particularly suitable for flexible and scalable automation implementations.

BMTools [119, 210] comprises an open-source code repository specifically engineered to enhance large linguistic model capabilities through integration with diversified toolsets while simultaneously providing a community-oriented developmental platform for the creation and dissemination of these auxiliary tools. This repository [119, 210] enables streamlined plugin development through simplified Python functional implementations and supports integration with external ChatGPT plugin architectures, establishing it as a significant resource for extending computational model operational capabilities. RestGPT [152] provides a framework for LLMs to interact with RESTful APIs through a structured pipeline of planning, selection, and execution, while xLAM [131] and Octopus-v4 [22] demonstrate advanced approaches for tool usage through specialized model architectures.

B.4 Open Source Models

GRANITE-20B-FUNCTIONCALLING [1] is an open-source function calling model from IBM based on multi-task learning that achieves state-of-the-art performance among open models on the Berkeley Function Calling Leaderboard and demonstrates strong generalization capabilities through training on seven fundamental function calling tasks. Additionally, Cohere Command [27] provides APIs such as Command R and Command R+ that facilitate integration with external tools and data sources, enhancing developer access to function-calling features. Mistral, an open-source project, has released the Mistral 7B model, enabling custom function definitions for use during inference [5, 58]. NexusRaven [154] presents the NexusRaven V2-13B model, an open-source LLM that excels in advanced function calling, often outperforming GPT-4 in cybersecurity tool and API invocation. Gorilla [113] OpenFunctions specializes in API interactions, with its 7B model fine-tuned on API documentation to produce accurate function calls from natural language prompts. The Fireworks FireFunction V1 model [3] builds upon the capabilities of the Mistral 7B model, achieving near GPT-4 quality in structured information generation and decision-making tasks. Lastly, Nous Hermes 2 Pro [130, 161] incorporates elements from both Mistral 7B and Llama 3 8B models to deliver exceptional performance in function calling and structured output generation, as evidenced by its high accuracy in evaluations.

B.5 Training Resources and Datasets

AgentInstruct [211], AgentOhana [215], and Lumos [206] represent comprehensive datasets and training pipelines that span multiple domains and environments, providing valuable resources for training function-calling capable models.

These products collectively contribute to the advancement of LLM capabilities by enabling more sophisticated interactions with external systems, enhancing autonomy, and supporting the development of complex AI applications across various domains.