ToolDev: Self-Improving Tool Learning via Decomposition and Evolution

Anonymous ACL submission

Abstract

The tool-using capability of large language models (LLMs) enables them to access up-to-date external information and handle complex tasks. Current approaches to enhancing this capability primarily rely on distilling advanced models by data synthesis. However, this method incurs significant costs associated with advanced model usage and often results in data compatibility issues, led by the high discrepancy in the knowledge scope between the advanced model and the target model. To address these challenges, we propose **ToolDev**, a self-improving framework for tool learning. First, we decompose the tool-learning objective into sub-tasks that enhance basic tool-making and tool-using abilities. Then, we introduce a self-evolving paradigm that allows lightweight models to self-improve, reducing reliance on advanced LLMs. Extensive experiments validate the effectiveness of our approach across models of varying scales and architectures.

1 Introduction

Large Language Models (LLMs) have achieved remarkable progress in natural language processing. However, they face significant limitations, including factual inaccuracies and challenges in accessing real-time information or executing actions. Enhancing their ability to use external tools—such as search engines (Schick et al., 2023; Nakano et al., 2021), APIs (Qin et al., 2023), and mathematical tools (Cobbe et al., 2021; He-Yueya et al., 2023)—is a promising solution. Tool integration not only grounds LLMs' outputs in reliable information but also expands their applicability to real-world scenarios requiring complex interactions.

Existing approaches to improve tool-utilization capabilities typically rely on distilling advanced models like GPT-4 or Claude 3.5 through data synthesis (Patil et al., 2023; Qin et al., 2023; Tang et al., 2023; Lin et al., 2024; Liu et al., 2024a). However, this strategy introduces three major challenges: 1) *Inference Cost.* Utilizing advanced models is prohibitively expensive, particularly when generating large-scale training datasets. 2) *Data Compatibility.* The synthesized data frequently exhibits distributional discrepancies, making it less compatible with the target model being fine-tuned. Specifically, unfamiliar samples—those introducing concepts outside the base model's knowledge scope—often lead to hallucinations (Hartmann et al., 2023; Kang et al., 2024). Consequently, target models tend to memorize the training data rather than generalize from it (Tirumala et al., 2022; Setlur et al., 2024), ultimately leading to suboptimal tool-utilization performance. 3) *Data Privacy.* In real-world applications, numerous user queries involve privacy constraints, prohibiting the synthesis using external advanced models. A promising alternative is self-evolution (Tao et al., 2024), where a model generates or refines its own training data, enabling iterative improvement without heavy reliance on external resources. Self-evolution has shown success in enhancing reasoning (Gulcehre et al., 2023; Singh et al., 2023; Huang et al., 2023) and code-generation (Jiang et al., 2023; Chen et al., 2024b) tasks through techniques like top-k sampling or nucleus sampling, where multiple solutions are generated, and correct ones are used for fine-tuning.

However, applying self-evolution to iterative improvement in tool-learning scenarios presents unique challenges. Tool-learning tasks typically consist of three components: user queries, candidate tools, and ground-truth tool invocations. Enhancing models with a diverse range of candidate tools during fine-tuning has been shown to improve their overall proficiency and zero-shot capabilities in tool utilization (Liu et al., 2024a). While lightweight, open-source LLMs demonstrate the ability to invoke tools from predefined candidate sets, they struggle to generate both novel tools and accurate invocations directly from user queries (Huang et al., 2024). This limitation makes the generation of high-quality, diverse training data a significant challenge.

To address the aforementioned challenges, we first **decompose** tool learning into several tool-related sub-tasks, enhancing the tool-making and tool-using abilities. Then we propose a **self-evolution** strategy specifically tailored for tool-learning, enabling the model to self-improve. The overall pipeline is termed as **ToolDev**. First, we identified that constructing tool documentation adaption tasks focused on tool definitions for post-trained models can effectively enhance the model's understanding of tools, thereby improving its tool-using and tool-generation capabilities. Subsequently, we decomposed the conventional tool-learning training objective, which typically concentrates solely on tool-using ability, into two tasks: *tool generation* and *tool invocation*. This approach strengthens the target model's ability to generate candidate tools based on a query, while simultaneously improving the accuracy of tool invocation, equipping the model with the foundational capabilities for self-evolution. Finally, after the aforementioned two-stage training, by providing new user queries, the target model iteratively generates candidate tools and corresponding invocations. This iterative process establishes a self-evolutionary mechanism that automatically enhances the model's tool-utilization performance over time. Our contributions can be summarized as follow:

- We propose ToolDev, the first self-evolutionary framework designed to enhance LLMs' tool-invocation capabilities, equipping lightweight models with self-evolving abilities.

- We propose the tool documentation adaption sub-task and decompose the tool-learning objective into tool generation and invocation tasks, demonstrating the task decomposition significantly improves tool-invocation performance.

- Through extensive experiments on LLMs of varying scales, we validate the effectiveness of our approach and provide insights into how self-evolution potential varies with model size.

2 Related Work

2.1 Tool Learning

The integration of external tools significantly enhances the capabilities of large language models (LLMs), enabling them to perform more specialized, accurate, and reliable problem-solving tasks (Qin et al., 2023). Existing methods for equipping LLMs with tool-use capabilities can be broadly categorized into two types: prompt-based approaches and tool-augmented tuning. Prompt-based methods enable LLMs to use tools by providing in-context examples and tool descriptions, bypassing the need for additional model training (Mialon et al., 2023; Hsieh et al., 2023; Ruan et al., 2023). A notable example is the ReAct framework (Yao et al., 2023), which allows LLMs to alternate between reasoning and executing actions to solve complex tasks. While tuning-free methods are lightweight and flexible, their performance is heavily reliant on the LLM's intrinsic capabilities, which limits their effectiveness for tasks requiring advanced tool utilization. In contrast, tool-augmented tuning methods directly enhance LLMs' tool-use capabilities through additional training (Qin et al., 2023; Schick et al., 2023; Patil et al., 2023; Tang et al., 2023; Liu et al., 2024b; Abdelaziz et al., 2024; Liu et al., 2024a; Lin et al., 2024). These methods typically involve fine-tuning LLMs to use external APIs and tools. However, a common limitation is the demand for high-quality data, which highly relies on data synthesis by an advanced model, such as GPT-4 or Claude-3.5. This generation process is not only resource-intensive but also incurs significant costs.

2.2 Self Evolution of LLMs

Self-evolution enables models to acquire and update knowledge autonomously, akin to human learning. For instance, the transition from AlphaGo (Silver et al., 2016) to AlphaZero (Silver et al., 2017) utilized a self-play mechanism to facilitate model evolution without reliance on labeled data. In the context of LLM self-evolution, research often focuses on two stages: task acquisition and solution generation. During the task acquisition stage, the target model generates new tasks. For example, Self-Instruct (Wang et al., 2022) enables models to autonomously generate new instructions as tasks, while Ada-Instruct (Cui and Wang, 2023) proposes an adaptive approach for task instruction generation. WizardLM (Xu

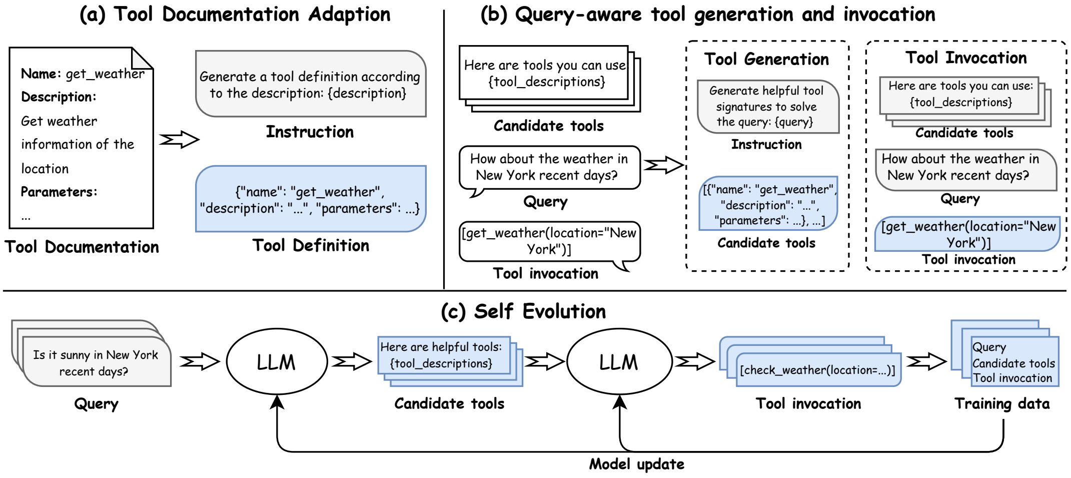

(a) Tool Documentation Adaption
Name: get_weather
Description:
Get weather information of the location
Parameters:
...
Tool Documentation
Generate a tool definition according to the description: {description}
Instruction
{"name": "get_weather", "description": "...", "parameters": ...}
Tool Definition
(b) Query-aware tool generation and invocation
Here are tools you can use {tool_descriptions}
Candidate tools
How about the weather in New York recent days?
Query
[get_weather(location="New York")]
Tool invocation
Tool Generation
Generate helpful tool signatures to solve the query: {query}
Instruction
[{"name": "get_weather", "description": "...", "parameters": ...}, ...]
Candidate tools
Tool Invocation
Here are tools you can use: {tool_descriptions}
Candidate tools
How about the weather in New York recent days?
Query
[get_weather(location="New York")]
Tool invocation
(c) Self Evolution
Is it sunny in New York recent days?
Query
LLM
Here are helpful tools: {tool_descriptions}
Candidate tools
LLM
[check_weather(location=...)]
Tool invocation
Query
Candidate tools
Tool invocation
Training data
Model update

Figure 1: Overall framework of the self-evolving paradigm. (a) Tool documentation adaption, aiming to enhance the understanding of tools; (b) Query-aware tool generation and invocation, equipping the model with self-evolving abilities; (c) Self-evolution, where the model first generates candidate tools and then generates tool invocation, forming the self-training data.

et al., 2023) introduces the Evol-Instruct method, which evolves instructions through both depth and breadth. In the solution generation stage, emphasis is placed on producing suitable answers for tasks. The STaR (Zelikman et al., 2024) framework incorporates the model's reasoning process and uses correct problem-solving steps as training data. REST (Gulcehre et al., 2023) and REST^{em} (Singh et al., 2023) employ sampling strategies to generate multiple trajectories and leverage a reward model to guide updates. Other approaches utilize both positive and negative sample pairs for preference-based training, such as DPO (Rafailov et al., 2024). Self-Reward (Yuan et al., 2024), for instance, constructs preference pairs by using the model itself as a reward model after solution generation. SPIN designates model-generated data as negative samples and labeled SFT data as positive samples. GRATH (Chen et al., 2024a) explicitly generates both positive and negative samples simultaneously, while Self-Contrast (Zhang et al., 2024b) compares differences between solutions and compiles these differences into a checklist for iterative refinement. In this work, we implement both task acquisition and solution generation, achieving completely autonomous evolution for LLMs.

3 Methodology

3.1 Task Definitions

Given the user query q and candidate tools $T = \{t_1, t_2, \cdots, t_N\}$, the goal of the tool invocation task is to select suitable tools and extract information as arguments A with the model parameters Θ:

$$A = [\cdots, (t_j, a_j), \cdots] = f(q, T, \Theta)$$

where t_j and a_j represent the j-th called tool and corresponding arguments, respectively. $f(\cdot)$ denotes the auto-regressive generation manner of LLMs. The training sample tailored for tool-using is usually formalized as a triplet of a user query, candidate tools and the ground-truth answers: $\langle q, T, A \rangle$.

The overall framework of ToolDev comprises three stages: tool documentation adaption, query-aware tool generation and innovation, and self-evolution, which is illustrated in Figure 1.

3.2 Tool Documentation Adaption

To enhance the LLM's capability on specific domains, continual pre-training on domain data is usually adopted as an effective method (Wu et al., 2023; Singhal et al., 2025). Drawing inspiration from this, we propose to train LLM on tool documentation for better tool understanding. This process enables the model to acquire a more in-depth understanding of the syntax, functionality, and constraints of specific tools, which can significantly improve its utility in real-world applications. Unlike general-purpose pre-training, this approach equips the model with domain-specific knowledge directly related to tool usage, reducing the gap between training data and deployment scenarios. Unlike other methods that continually pre-train a "base"-series LLM without any post-training such as instruction tuning, we design a adaption task for "Instruct"-series models, thereby keeping the instruct-following ability obtained in post-training.

Specifically, given the documentation of a tool: $t_i = \langle name, description, parameters \rangle$, we construct an instruction x_{t_i} to ask the model to complete tool definitions according to tool description, such as "*Generate a tool signature according to the description: {description}*". Then the training procedure aligns with an instruction tuning:

$$\min_{\Theta} \ell \left(f \left(x_{t_i}, \Theta \right), t_i \right) \tag{1}$$

where $\ell(\cdot)$ is the loss function to align the model's prediction with the tool's documentation. The task enables the LLM to be familiar with the format of tool definitions, building the fundamental ability to construct tools for coming queries.

3.3 Query-Aware Tool Generation and Invocation

Unlike question-answering or coding tasks, which typically involve only queries and answers, tool invocation data often requires not only the user's query but also a set of candidate tools. Existing studies generally rely on a finite set of candidate tools sampled from a fixed pool. However, this approach overlooks a critical issue: when models are applied to new scenarios with unseen candidate tools and queries, the accuracy of tool invocation tends to suffer significantly. Therefore, a tool invocation model with self-evolution capabilities should ideally possess the ability to expand its training set of tools autonomously.

To address this challenge, we decompose the tool learning data into two sub-tasks: query-aware tool generation and tool invocation. In existing tool-learning training, the focus is typically on the tool invocation, i.e., given a query and a set of candidate tools, aligning the model's predicted tool invocation with the ground truth A:

$$\min_{\Theta} \ell \left(f \left(q, T, \Theta \right), A \right) \tag{2}$$

Contrastively, our decomposition introduces an additional task during training: generating query-relevant candidate tools based on the given query. This aims to both enhance the model's understanding of the relationship between queries and tools and equip it with the ability to autonomously generate relevant candidate tools, thereby preparing it for self-evolution. Similar to tool documentation adaption, given a query q, we convert it into an instruction format x_q, such as "*Generate candidate tools related to the query: {query}*." The training objective is then to generate the corresponding candidate tools T:

$$\min_{\Theta} \ell \left(f \left(x_q, \Theta \right), T \right) \tag{3}$$

By training with this objective, the model becomes capable of generating candidate tools for incoming queries, transcending the limitations of a finite tool set, thereby opening the door to self-evolution.

3.4 Self-Evolution

After training through the first two stages, the model acquires the foundational capabilities for self-evolution: the ability to generate candidate tools based on a given query, and the ability to invoke tools based on the query and its corresponding candidate tools. When confronted with a new query, the model can autonomously generate new tool invocation training data. The self-evolution process is primarily composed of three steps: candidate tool generation, tool invocation generation, and model updating, illustrated in Figure 1(c).

Candidate tool generation. Upon a new query q is collected, it is first reformulated into an instruction x_q to guide the model in generating a set of candidate tools $\tilde{T}$ relevant to the query:

$$\tilde{T} = f(x_q, \Theta^{(i)}) \tag{4}$$

To ensure the correctness of the format of generated tools, we adopt a rule checker to filter out those problematic samples, such as missing argument descriptions or JSON-unparsable.

Tool invocation generation. After the generation of candidate tools, the model is then prompted to generate tool calls $\tilde{A}$ to solve the query with generated tools $\tilde{T}$. To improve the correctness of the generated solution, we obtain multiple solutions via the top-k sampling strategy and then majority voting is applied to select the answer as the ground truth. The sampling and voting process, termed as self-consistency decoding (Wang et al., 2023), has been validated as an effective method to improve the performance of LLMs. From the perspective of self-rewarding, it assigns a positive reward to the solution with higher confidence:

$$\tilde{A} = \text{majority\_vote}(f_{sampling}(q, \tilde{T}, \Theta^{(i)}) \tag{5}$$

Table 1: Accuracy performance comparison on BFCL leaderboard. The top 20 models are listed for comparison. The models are sorted according to the overall score. FC denotes the model is tailored for functional calling.

Model	Non-Live				Live				Overall		
	Simple	*Multi*	*Parallel*	*Parallel Multi*	*Simple*	*Multi*	*Parallel*	*Parallel Multi*	*Non-live*	*Live*	*Overall*
GPT-4-turbo (Prompt)	82.25	94.50	95.00	93.50	78.68	83.12	81.25	75.00	91.31	82.09	86.70
xLAM-8x22b-r (FC)	77.00	95.50	92.50	94.00	70.93	77.72	75.00	75.00	89.75	76.33	83.04
ToolDev(FC)	80.17	97.50	93.50	87.50	70.16	76.37	81.25	75.00	89.67	75.20	82.44
Llama-3-70B-Instruct (Prompt)	75.83	94.50	91.50	87.00	69.77	78.01	75.00	66.67	87.21	76.18	81.69
mistral-large (FC)	57.50	94.00	93.00	92.00	79.07	78.88	87.50	75.00	84.12	78.95	81.54
xLAM-8x7b-r (FC)	77.25	95.50	92.00	89.00	68.99	76.18	50.00	75.00	88.44	74.46	81.45
ToolACE-8B (FC)	80.58	95.00	91.00	90.50	62.79	74.25	81.25	75.00	89.27	72.13	80.70
GPT-4o-mini (Prompt)	79.67	89.50	89.00	88.00	72.09	73.77	81.25	70.83	86.54	73.48	80.01
GPT-3.5-Turbo (FC)	74.08	93.00	87.50	83.50	65.50	74.16	56.25	54.17	84.52	71.91	78.22
FireFunction-v2 (FC)	78.83	92.00	91.00	81.00	69.38	70.97	56.25	54.17	85.71	70.18	77.95
GPT-4-turbo (FC)	60.58	91.00	90.00	89.00	67.83	74.45	75.00	62.50	82.65	72.96	77.81
GPT-4o (FC)	73.58	92.50	91.50	84.50	67.83	69.43	75.00	66.67	85.52	69.14	77.33
GPT-4o-mini (FC)	67.83	90.50	89.50	83.50	67.83	69.82	81.25	70.83	82.83	69.59	76.21
Gorilla-OpenFunctions-v2 (FC)	77.67	95.00	89.00	87.50	63.95	63.93	62.50	45.83	87.29	63.59	75.44
xLAM-7b-fc-r (FC)	77.33	92.50	91.50	86.00	63.57	63.36	56.25	50.00	86.83	63.08	74.95
Open-Mistral-Nemo (FC)	60.92	92.00	85.50	85.50	68.22	67.98	75.00	62.50	80.98	68.01	74.50
GPT-4o (Prompt)	64.08	86.50	88.00	85.00	67.44	67.21	56.25	58.33	80.90	66.96	73.93
Gemini-1.5-Flash-Preview (FC)	65.42	94.50	71.50	77.00	62.79	72.61	56.25	54.17	77.10	70.18	73.64
Claude-3.5-Sonnet (FC)	75.42	93.50	62.00	50.50	72.48	70.68	68.75	75.00	70.35	71.08	70.72
Gemini-1.5-Pro-Preview (FC)	50.17	89.50	83.50	79.00	60.08	66.35	75.00	54.17	75.54	65.02	70.28
o1-mini (Prompt)	68.92	89.00	73.50	70.50	62.79	65.09	68.75	58.33	75.48	64.57	70.02

where $\mathrm{majority\_vote}(\cdot)$ denotes select the solutions with the most votes and $f_{sampling}$ denotes the sampling-based decoding strategy, which is implemented as top-k sampling in our experiments. Also, a rule checker is applied to filter out those samples with unreasonable solutions, such as calling hallucinating tools or arguments and filling arguments with wrong types.

Model updating. For each incoming query, candidate tool generation and tool invocation generation can turn the query q to a complete tool-using triplet $\langle q, \tilde{T}, \tilde{A} \rangle$. Then a new training set can be collected after repeating the first two steps on all queries, where the model can be trained with two types of objectives: query-aware tool generation and invocation, as proposed in Section 3.3:

$$\Theta^{(i+1)} = \min_{\Theta} \ell\big(f(q, \tilde{T}, \Theta), \tilde{A}\big) + \ell\big(f\left(x_q, \Theta\right), \tilde{T}\big) \qquad (6)$$

4 Experiments

4.1 Experimental Settings

Datasets Construction. In the first phase of training, we sampled a subset of data from ToolACE (Liu et al., 2024a), comprising a total of 26,522 tools, to perform tool documentation adaption on the model. Subsequently, in the second phase, we utilized a dataset containing 20,000 synthesized tool invocation samples generated by GPT-4 [1] for further fine-tuning. In each subsequent self-evolution round, the model self-generates training data by processing 10,000 incoming queries. We set the max round of self-evolution as 3 and the best results (may not be at the third round) are adopted in Table 1. Note that the overall training utilizes 20,000 synthesized query-solution pairs and a maximum of 30,000 queries in total.

Benchmark and Evaluation. To evaluate the model's tool invocation capabilities, we selected the Berkeley Function Call Leaderboard (BFCL) (Yan et al., 2024), a widely recognized benchmark, as the evaluation framework. BFCL consists of two subsets: Non-live and Live, representing synthetic test cases and real-world scenarios, respectively [2]. Both non-live and live subsets comprise four types of test examples: simple, multiple, parallel, and parallel multiple. Simple

[1] `https://chatgpt.com`

[2] We focused exclusively on single-turn tool invocation AST data, as these test cases exhibit higher stability and reliability, whereas other cases tend to have significant variability and lower reliability.

Table 2: Results on other tool learning benchmarks. **Bold** and underline results represent the 1st and the 2nd best results.

Model	APIBank	T-Eval
GPT-4-turbo	63.39	**87.50**
Llama-3.1-8B-Instruct	54.11	76.60
ToolDev	**67.82**	77.03

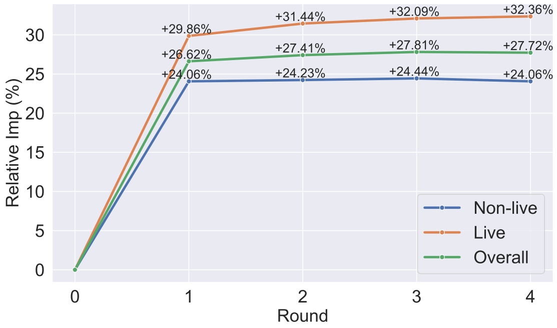


Figure 2: Relative Improvements of Self-Evolution. The backbone model is LLaMA-3.1-8B-Instruct.

and multiple examples both involve only one invoked tool, while there are multiple candidate tools in multiple examples. Parallel (or Parallel multiple) examples require invoking multiple tools from one (or multiple) candidate tool(s). The evaluation metrics for each subset are accuracy-based, and for certain categories, the scores are computed as the average of subcategory scores. To further validate the efficiency of our method, we evaluate ToolDev on another two tool-calling benchmarks: API-Bank (Li et al., 2023) and T-Eval (Chen et al., 2024c). The details of those benchmarks are reported in Appendix A.

Implementation Details. We employed the LLaMA3.1-8B-Instruct (AI@Meta, 2024) as the base model for training. Due to resource constraints, the parameter-efficient training technique, LoRA (Hu et al., 2022), is conducted on 8 Nvidia V100-32GB GPUs. All model modules are enabled for LoRA fine-tuning, with the LoRA rank set to 16 and alpha set to 32. The training processes utilize a global batch size of 64 and a learning rate of 10^{-4} with a commonly used cosine learning rate scheduler, where the warmup ratio is set as 0.1. The prompts in each stage are illustrated in Appendix B. More details are provided in Appendix A.

4.2 Main Results

To demonstrate the superiority of the model performance under the training framework we propose, we compare the tool invocation accuracy of the top 20 models in the BFCL leaderboard [3]. And we compare the GPT-4-turbo and Llama3.1-8B-Instruct on other two benchmarks. The results are shown in Table 1 and Table 2, where we can have the following observations:

First, ToolDev, trained using our proposed self-evolution framework, achieves high accuracy at the 8B model scale only, surpassing larger models such as LLaMA-3-70B-Instruct, several closed-source models like Claude, Gemini, and GPT-4o, and models that are specifically fine-tuned for tool invocation. The performance of ToolDev is on par with that of large MoE models like xLAM-8x22B-r. Besides, ToolDev shows significant improvements on another two benchmarks compared with Llama-3.1-8B-Instruct. This is attributed to the effectiveness of our training framework.

Second, compared to ToolACE-8B that is finetuned from LLaMA-3.1-8B-Instruct as well, ToolDev still demonstrates further improvements, achieving higher scores in BFCL. Our new training framework enables ToolDev to achieve significant improvements with only a minimal amount (20,000) of labeled training data, resulting in reduced training data costs while increasing the data utilization efficiency. Additionally, it fully leverages the model's capability to generate its own data, showcasing that the self-evolution process is as effective as data synthesis with advanced models. This suggests that self-evolution is highly effective for tool-invocation tasks and may become a more efficient approach for data acquisition in the future.

Furthermore, ToolDev shows a more significant improvement on the more challenging Live subset than on the Non-live subset compared to ToolACE-8B. In the BFCL test set, the candidate tools in the Live category are user-contributed, which increases their authenticity and diversity compared to the Non-live category, making the queries in the Live subset more complex. ToolDev achieves a larger gain in this more difficult subset, which can be attributed to our tool documentation adaption and query-aware tool generation auxiliary tasks, which improve data utilization. Additionally, the model's self-evolution process generates high-quality train-

[3] The data is sourced from the BFCL leaderboard update on 2024-09-20, referenced from https://github.com/ShishirPatil/gorilla/blob/e82d4246bec26276cceade9c710df92b9d83420a/data_combined_Sep_20_2024.csv

Table 3: Ablation study on the proposed training objectives. **Invo**. and **Gen.** represent the tool invocation and tool generation task, respectively. "w. Adaption" represents the model is trained successively from Adaption.

Model	Non-live	Live	Overall
Raw	72.06	56.93	64.50
Adaption	73.54	57.52	65.53
Invo.	88.96	72.73	80.85
Invo. w. Adaption	89.08	73.03	81.06
Invo.+Gen. w. Adaption	**89.40**	**73.93**	**81.67**

ing data that is appropriately challenging, rather than simplistic or trivial.

4.3 Performance of Self-Evolution

To evaluate the effectiveness of the model's self-evolution mechanism, we conducted three rounds of self-training using the LLaMA-3.1-8B-Instruct model after it had undergone pre-training phases including Tool Documentation Adaption and Query-Aware Tool Generation and Invocation. In each round of evolution, the model processed 10,000 queries through a two-step generation, producing corresponding candidate tools and tool invocations, as detailed in Section 3.4. The results of each iterative step are presented in Figure 2.

It is evident that the scores for Non-live, Live, and Overall metrics consistently improve across iterations, indicating that the model successfully generates informative training data tailored to its current state. Notably, we observe a more pronounced improvement in the more challenging Live scores, suggesting that the data samples generated during self-evolution are appropriately challenging and contain substantial informational value. Additionally, we find that the performance gains from self-evolution diminish as the number of iterations increases. This aligns with conclusions drawn in prior study (Chen et al., 2024d), leading to a hypothesize: the self-iteration process gradually enhances the model's confidence in generating accurate tool invocations, and once the model's confidence becomes sufficiently high, the self-generated data contributes less to further improvements.

4.4 Ablation Study

To validate the effectiveness of the training objectives proposed at each stage, we conducted an ablation study to evaluate various variants. Specifically, we compared the following variants:

Raw: The raw model without extra post training.

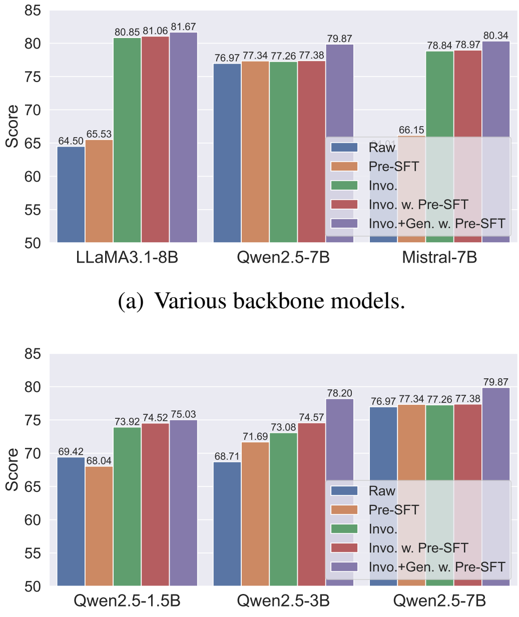
Adaption: The model undergoes only the first stage of Tool Documentation Adaption training, without any labeled tool invocation data.

Invo.: The model is trained exclusively on the tool invocation portion of the data, using the training objective in Equation 2.

Invo. w. Adaption: The model first undergoes Tool Documentation Adaption training, followed by training with the tool invocation objective.

Invo.+Gen. w. Adaption: The model first undergoes Tool Documentation Adaption training, then trains both the tool invocation and tool generation objectives simultaneously, optimizing the training objectives in Equation 2 and Equation 3.

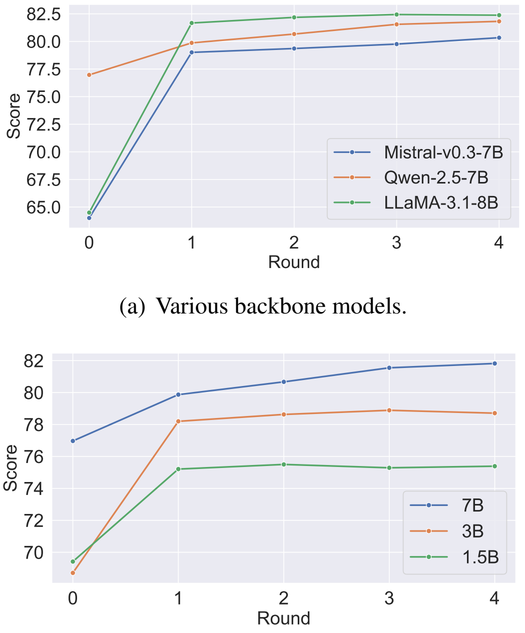
The evaluation results for each variant are shown in Table 3. First, the **Adaption** model shows improvement compared to the Raw model, and **Invo. w. Adaption** outperforms **Invo.**, indicating that Tool Documentation Adaption contributes to enhancing the model's understanding of tool definitions and syntax, thereby improving its tool invocation capability. Furthermore, **Invo.+Gen. w. Adaption** demonstrates a clear advantage among the variants, suggesting that the tool generation task, as an advanced tool-related capability, significantly aids in enhancing the model's tool-related performance.



(a) Various backbone models.

(b) Various scales of models.

Figure 3: Ablation study of different training objectives on various models.



(a) Various backbone models.

(b) Various scales of models.

Figure 4: Effects of self-evolution on various models.

4.5 Effectiveness on various models

To validate the effectiveness and generality of our proposed framework, we conducted experiments on various models, including models of different parameter scales within the same series and models of similar parameter scales across different series. Specifically, we trained the Qwen2.5-Instruct (Qwen-Team, 2024) series models with parameter sizes of 1.5B, 3B, and 7B, as well as the Mistral-v0.3-7B-Instruct (Mistral-AI, 2024) and LLaMA-3.1-8B-Instruct models. We evaluated the effectiveness of the proposed training objectives at each stage and the efficacy of the model's self-evolution mechanism.

Effectiveness of training objectives. The results of our proposed training objectives across various models are shown in Figure 3(a). As observed, Invo.+Gen. w. Adaption significantly outperforms all other variants, further validating the effectiveness of the proposed Adaption and Generation tasks. Additionally, due to the varying initial tool invocation capabilities of different model backbones, the improvements achieved by Adaption and the invocation tasks differ across models. For instance, Qwen2.5-7B, which exhibits a relatively strong initial tool invocation capability and a better understanding of tools, shows only marginal gains from the Adaption and tool invocation tasks. In contrast, the improvements are more pronounced for LLaMA-3.1-8B and Mistral-v0.3-7B.

As the model size increases, the effects of our proposed training strategy exhibit certain differences, as illustrated in Figure 3(b). For smaller models, such as 1.8B model, the weaker instruction-following capabilities result in a tendency to generate tools rather than invoke them when only Tool Documentation Adaption is applied, leading to a decline in tool invocation scores. Additionally, due to the limited number of parameters, the improvements achieved through multitask training in **Invo.+Gen. w. Adaption** are comparatively smaller. In this case, the tool generation task poses a greater challenge for smaller models, making it more difficult for them to generalize effectively.

Effectiveness of self-evolution. The self-evolution performance of different models is illustrated in Figure 4. Our findings are as follows: First, for models with 7-8B parameters, the self-evolution results are consistently positive, aligning with the conclusions drawn in Section 4.3. Second, larger models exhibit greater potential for self-evolution. For instance, the 7B models show improvement across all evolution iterations, whereas the 3B models display a slight downward trend in the final iteration, and the 1.5B models exhibit a convergence trend with fluctuations in the last two iterations. This behavior may be attributed to smaller models being more prone to overfitting the training data distribution after 1-2 iterations, which reduces the diversity of the subsequently self-generated data and limits further improvements.

5 Conclusion

In this work, we proposed a training framework designed to enhance the tool invocation capabilities of large language models (LLMs), enabling effective self-evolution in tool-related tasks. The training algorithm begins with a Tool Documentation Adaption task to strengthen the model's understanding of tools. Subsequently, we decompose tool-learning data into query-aware tool generation and invocation sub-tasks, empowering the model with the ability to generate tools tailored to specific queries. Building on this foundation, the model can iteratively improve itself by generating data based on given queries. Experimental results demonstrate that our training approach endows the model with self-evolution capabilities, achieves superior tool invocation accuracy compared to all other models of similar scale, and validates the generality of the proposed method across various models.

Limitations

First, our experiments were conducted on models up to 7B due to resource constraints, leaving the self-evolution performance of larger models (e.g., 14B, 32B) unexplored. Given existing results, larger models are likely to generate higher-quality data, potentially enhancing self-evolution. Additionally, this work focuses on tool invocation accuracy—selecting the correct tool and providing precise parameters—but does not address retrieving tools from a large-scale tool pool, an important avenue for future research.

References

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*.

AI@Meta. 2024. Llama 3 model card.

Weixin Chen, Dawn Song, and Bo Li. 2024a. Grath: gradual self-truthifying for large language models. *arXiv preprint arXiv:2401.12292*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024b. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024c. 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)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024d. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Wanyun Cui and Qianle Wang. 2023. Ada-instruct: Adapting instruction generators for complex reasoning. *arXiv preprint arXiv:2310.04484*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. 2023. Sok: Memorization in general-purpose large language models. *arXiv preprint arXiv:2310.18362*.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *Preprint*, arXiv:2308.00675.

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

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore. Association for Computational Linguistics.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv preprint arXiv:2401.17167*.

Shuyang Jiang, Yuhao Wang, and Yu Wang. 2023. Self-evolve: A code evolution framework via large language models. *arXiv preprint arXiv:2306.02907*.

Katie Kang, Eric Wallace, Claire Tomlin, Aviral Kumar, and Sergey Levine. 2024. Unfamiliar finetuning examples control how language models hallucinate. *arXiv preprint arXiv:2403.05612*.

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

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

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*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2024a. Toolace: Winning the points of llm function calling. *Preprint*, arXiv:2409.00920.

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

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey. *Preprint*, arXiv:2302.07842.

Mistral-AI. 2024. Mistral-7b-instruct-v0.3.

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*.

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*.

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*.

Qwen-Team. 2024. Qwen2.5: A party of foundation models.

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.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Large language model-based ai agents for task planning and tool usage. *Preprint*, arXiv:2308.03427.

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. *Preprint*, arXiv:2302.04761.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *arXiv preprint arXiv:2406.14532*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*.

Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. 2025. Toward expert-level medical question answering with large language models. *Nature Medicine*, pages 1–8.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*.

Kushal Tirumala, Aram Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. 2022. Memorization without overfitting: Analyzing the training dynamics of large language models. *Advances in Neural Information Processing Systems*, 35:38274–38290.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

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

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Eric Zelikman, YH Wu, Jesse Mu, and Noah D Goodman. 2024. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2024a. xlam: A family of large action models to empower ai agent systems. *Preprint*, arXiv:2409.03215.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024b. Self-contrast: Better reflection through inconsistent solving perspectives. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3602–3622, Bangkok, Thailand. Association for Computational Linguistics.

A Experimental Details

A.1 Benchmark Details

Berkeley Function-Calling Leaderboard (BFCL). The BFCL (Yan et al., 2024) benchmark consists of Non-Live and Live categories, where each category comprises single, multiple, parallel and parallel multiple samples.

- Single: A single function evaluation represents the most straightforward yet commonly encountered format, where the user supplies a single JSON function document, and exactly one function call is invoked.

- Multiple: The multiple function category involves a user query that triggers a single function call selected from among 2 to 4 available JSON function documents. The model must be capable of determining the most appropriate function to invoke based on the context provided by the user.

- Parallel: A parallel function entails the simultaneous invocation of multiple function calls in response to a single user query. The model must determine the number of required function calls, with the user's query potentially consisting of a single sentence or multiple sentences.

- Parallel Multiple: Parallel multiple functions combine the concepts of parallel function and multiple function. In this scenario, the model is provided with several function documents, and each corresponding function call may be invoked zero or more times.

API-Bank. API-Bank (Li et al., 2023) is a benchmark designed to evaluate and enhance the tool-augmented capabilities of LLMs, too. It features a runnable evaluation system with 73 API tools and an annotated dataset of 314 dialogues containing 753 API calls, used to assess LLMs' ability to plan, retrieve, and call APIs. In this work, we mainly focus on the tool-invocation task, averaging the correctness of *Call* and *Retrieve+Call* in API-Bank as the overall score.

T-Eval. T-Eval (Chen et al., 2024c) takes several abilities helpful for tool invocation into evaluation, including the instruction following, planning, reasoning, retrieval, understanding, and review. In this work, we average scores of three tool-invocation task as the overall score: planning, retrieval and understanding, where planning and retrieval represent tool selection and understanding represents parameters filling.

A.2 Baselines

We have selected top 20 LLMs on BFCL as baseline models as they show advantaged tool-calling performance, including closed models and open-sourced models. Closed models include GPT-series from OpenAI, Gemini-series from Google and Claude-series from Anthropic. Open-sourced models include general LLMs, such as LLaMA-3-series and mistral-series, and tool-augumented LLMs, such as xLAM-series (Zhang et al., 2024a) and OpenFunctions-series (Patil et al., 2023). For API-Bank and T-Eval, we have compared the state-of-the-art GPT-4-turbo and the LLaMA-3.1-Instruct-8B.

A.3 Implementation Details

In the self-evolution stage, we leverage the target model to generate candidate tools and tool invocations by itself. vLLM framework (Kwon et al., 2023) is used to accelerate the generation process. For the generation of the candidate tools, we set the temperature of generation as 1.0, aiming to enhance the diversity of generated tools. For the generation of tool invocation, we utilize the self-consistency decoding strategy, generating 5 solutions for each sample and selecting the solution with the most votes as the final solution.

B Prompts

In this section, we illustrate all prompts used in the training framework, including the tool invocation task(Figure 5), tool documentation pre-SFT task(Figure 6) and query-aware tool generation task(Figure 7). In each task, an example comprising of the input prompt and the expected output is detailed.

[SYSTEM]
You are a helpful assistant. You can call various user-defined tools to solve users' problems.

# Tools You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>

```
{
  "name": "Division API",
  "description": "Divide two time series and return the result.",
  "arguments": {
    "type": "dict",
    "properties": {
      "interval": {
        "description": "Interval between two consecutive points in time series.
         Supports: `1min`, `5min`, `15min`, `30min`, `45min`, `1h`, `2h`, `4h`, `1day`, `1week`",
        "type": "string"
      },
      "symbol": {
        "description": "Instrument symbol, can be any equity, index, ETF, forex or cryptocurrency.
          E.g. `AAPL`, `EUR/USD`, `ETH/BTC`, ...",
        "type": "string"
      },
      ...
    "required": ["interval", "symbol"]
  }
  ...
}
```

</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
"name": <function-name>, "arguments": <args-json-object>
</tool_call>

[USER]
Could you provide the ratio of the high to the close prices over different intervals for the AAPL symbol? I'd need 1min, 15min and 1h intervals. Make sure to use JSON format for the results.

[ASSISTANT]

```
<tool_call>
{
 "name": "Division API",
 "arguments": {
   "interval": "1min", "symbol": "AAPL", "series_type_1": "high",
   "series_type_2": "close", "format": "json"
  }
}
</tool_call>
<tool_call>
{
  "name": "Division API",
  "arguments": {
    "interval": "15min", "symbol": "AAPL", "series_type_1": "high",
    "series_type_2": "close", "format": "json"
  }
}
</tool_call>
<tool_call>
{
  "name": "Division API",
  "arguments": {
    "interval": "1h", "symbol": "AAPL", "series_type_1": "high",
    "series_type_2": "close", "format": "json"
  }
}
</tool_call>
```

Figure 5: Illustration of the tool invocation task. The blue part represents the expected output.

[SYSTEM]
You are a helpful assistant. You are a helpful assistant. You are skilled in generating tool signatures. You can generate high-quality tool signature according to the instruction.

Output the tool signature within <tools></tools> XML tags:

```
<tools>
{
  "name": "xxx",
  "description": "xxx",
  "arguments": {
    "type": "dict",
    "properties": {
      "arg1": {
        "type": "xxx",
        "description": "xxx"
      },
      "arg2": {...}
    },
    "required": ["arg1", ...]
  },
  "results": {
    "type": "dict",
    "properties": {
      "result_1": {"type": "xxx", "description": "xxx"}, "result_2": {...}}
  }
}
</tools>
```

[USER]
Generate a tool definition according to the description: Divide two time series and return the result.

[ASSISTANT]

```
<tools>
{
  "name": "Division API",
  "description": "Divide two time series and return the result.",
  "arguments": {
    "type": "dict",
    "properties": {
      "interval": {
        "description": "Interval between two consecutive points in time series.
         Supports: `1min`, `5min`, `15min`, `30min`, `45min`, `1h`, `2h`, `4h`, `1day`, `1week`",
        "type": "string"
      },
      "symbol": {
        "description": "Instrument symbol, can be any equity, index, ETF, forex or cryptocurrency.
          E.g. `AAPL`, `EUR/USD`, `ETH/BTC`, ...",
        "type": "string"
      },
      ...
    "required": ["interval", "symbol"]
  }
  ...
}
</tools>
```

Figure 6: Illustration of the tool documentation pre-SFT task. The blue part represents the expected output.

[SYSTEM]

You are a helpful assistant. You are a helpful assistant. You are skilled in generating tool signatures. You can generate high-quality tool signature according to the instruction.

Output the tool signature within <tools></tools> XML tags:

```
<tools>
{
  "name": "xxx",
  "description": "xxx",
  "arguments": {
    "type": "dict",
    "properties": {
      "arg1": {
        "type": "xxx",
        "description": "xxx"
      },
      "arg2": {...}
    },
    "required": ["arg1", ...]
  },
  "results": {
    "type": "dict",
    "properties": {
      "result_1": {"type": "xxx", "description": "xxx"}, "result_2": {...}}
  }
}
</tools>
```

[USER]

Generate tools to solve the query: Could you provide the ratio of the high to the close prices over different intervals for the AAPL symbol? I'd need 1min, 15min and 1h intervals. Make sure to use JSON format for the results.

[ASSISTANT]

```
<tools>
{
  "name": "Division API",
  "description": "Divide two time series and return the result.",
  "arguments": {
    "type": "dict",
    "properties": {
      "interval": {
        "description": "Interval between two consecutive points in time series.
         Supports: `1min`, `5min`, `15min`, `30min`, `45min`, `1h`, `2h`, `4h`, `1day`, `1week`",
        "type": "string"
      },
      "symbol": {
        "description": "Instrument symbol, can be any equity, index, ETF, forex or cryptocurrency.
          E.g. `AAPL`, `EUR/USD`, `ETH/BTC`, ...",
        "type": "string"
      },
      ...
    "required": ["interval", "symbol"]
  }
  ...
}
</tools>
```

Figure 7: Illustration of the query-aware tool generation task. The blue part represents the expected output.