TOOLVERIFIER: Generalization to New Tools via Self-Verification

Dheeraj Mekala[*,◇,♠], **Jason Weston**[◇], **Jack Lanchantin**[◇],
Roberta Raileanu[◇], **Maria Lomeli**[◇], **Jingbo Shang**[♠], **Jane Dwivedi-Yu**[◇]
[◇] Meta [♠] University of California San Diego
[♠]{dmekala, jshang}@ucsd.edu
[◇]{jase, jacklanchantin, raileanu, marialomeli, janeyu}@meta.com

Abstract

Teaching language models to use tools is an important milestone towards building general assistants, but remains an open problem. While there has been significant progress on learning to use specific tools via fine-tuning, language models still struggle with learning how to robustly use new tools from only a few demonstrations. In this work we introduce a self-verification method which distinguishes between close candidates by self-asking contrastive questions during (1) tool selection; and (2) parameter generation. We construct synthetic, high-quality, self-generated data for this goal using Llama-2 70B, which we intend to release publicly. Extensive experiments on 4 tasks from the ToolBench benchmark, consisting of 17 unseen tools, demonstrate an average improvement of 22% over few-shot baselines, even in scenarios where the distinctions between candidate tools are finely nuanced.

1 Introduction

Incorporating external tools into large language models (LLMs) enhances their real-world applicability (Schick et al., 2023; Shen et al., 2023; Song et al., 2023). Many tools exist in the form of APIs (Xu et al., 2023b; Tang et al., 2023; Hsieh et al., 2023; Schick et al., 2023; Qin et al., 2023), machine learning models (Shen et al., 2023; Patil et al., 2023), and other functions (Gou et al., 2023). Nevertheless, the evolving landscape of existing tools and APIs, marked by frequent parameter updates and the daily introduction of new tools, poses a challenge for generalization. LLMs must quickly adapt to these changes and generalize to previously unseen tools without additional fine-tuning or extensive human input.

Several recent studies enable tool usage by fine-tuning LLMs on real (Schick et al., 2023; Qin et al., 2023; Patil et al., 2023) or synthetic tools (Tang et al., 2023), equipping them to effectively utilize tools present in the training data with a high success rate. Currently, the integration of unseen tools into LLMs relies on providing them with few-shot demonstrations that contain examples of user instructions and corresponding tool calls (Patil et al., 2023; Tang et al., 2023). However, these prompting-based approaches still struggle to accurately generate a complete tool call from a set of unseen tools.

To address these challenges, we propose TOOLVERIFIER, a self-verification method tailored for tool-use scenarios, capable of discerning between candidate tools and their respective parameters through verification questions. To achieve this, we decompose the tool call generation task into two distinct sub-tasks: (1) *tool selection*, given a user instruction, the most suitable tool is selected from a library of options, and (2) *parameter generation*, the appropriate parameters for the selected tool are then generated. Crucially, we propose verification for each sub-task, to both improve sensitivity and to curb error propagation. Figure 1 shows an overview of each sub-task.

In the tool selection stage, our model must choose one tool among multiple options, given only the description of the tool. To facilitate learning how to choose the appropriate tool, we curate a high-quality, model-generated, synthetic training dataset containing tools, their descriptions, and user instructions.[1] This dataset comprises 173 synthetic tools with corresponding descriptions, 555 samples, each involving reasoning about the tool's usage. We then use this dataset to fine-tune a Llama-2 70B model (Touvron et al., 2023b) to select the correct tool for an instruction given only a set of tool names and their descriptions, allowing the model at test

[*] Work done during an internship at Meta.

[1]The dataset is available at https://github.com/facebookresearch/ToolVerifier

Findings of the Association for Computational Linguistics: EACL 2024, pages 5026–5041

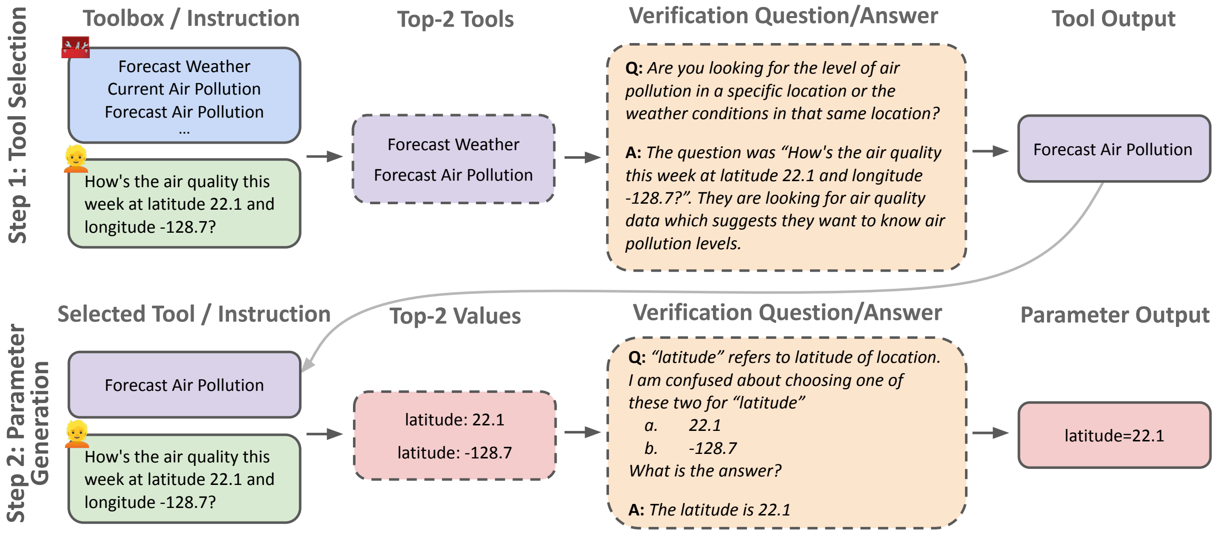

Figure 1: Overview of TOOLVERIFIER. Starting with a candidate tool list and a user instruction, TOOLVERIFIER initially identifies the top two tools. Subsequently, it generates a verification question by contrasting the selected tools and answers it. Finally, this information is appended to the context, leading to the final tool choice. The parameter generation follows a similar pipeline, wherein two candidate values are obtained for each parameter (*latitude* in the above figure). Subsequently, the verification question is used to finalize the parameter value.

time to select from tools never seen during training. After the tool is selected, parameters are generated for the selected tool call, which is achieved through few-shot prompting with demonstrations corresponding to the chosen tool.

Self-verification is used at each step to reduce error propagation and enhance overall performance. As shown in Figure 1, for tool selection verification, we extract the top two predictions from the fine-tuned model. A verification question is then generated *contrasting the two options* via 0-shot prompting, enabling the model to focus on a fine-grained decision where the answer aids in selecting one tool from the top two predictions. The model answers the question, and the context is updated by appending this answer to the user instruction, to guide tool selection. A similar approach is adopted for verifying the parameter generation.

We evaluate our approach on 4 tasks from the publicly available ToolBench benchmark which tests generalization to 17 unseen real-life APIs. TOOLVERIFIER demonstrates a noteworthy 22% improvement over few-shot prompting baselines. The proposed self-verification mechanism contributes an improvement of 8%, underscoring its pivotal role in boosting overall performance.

2 TOOLVERIFIER

TOOLVERIFIER chooses and calls a tool given a user instruction. It consists of the following steps:

1. Tool selection & verification – *selecting the tool from a library of tools.*
2. Parameter generation & verification – *generating the parameters for the tool call.*

For step (1) we generate synthetic data consisting of a library of tools, (instruction, tool) pairs, and reasoning notes explaining the correct choice of tool, see Figure 2. Fine-tuning on this data provides improved tool selection performance, even on new sets of tools. The selection process is then refined by verifying the choice between the top two competing choices by asking and answering *contrastive verification questions*, see Figure 3.

For step (2) we use few-shot prompting given demonstrations of the actual tool. We again verify two competing likely generations.

2.1 Tool Selection Dataset Generation

Our first goal is to train a language model capable of selecting an appropriate tool for a given user instruction by reasoning about a candidate list of tools solely based on their names and descriptions. We intentionally exclude demonstrations for tool selection in our approach to handle a larger set of tools in one go, using only their names and descriptions. In this section, we elaborate on the process of creating the training dataset for training such a tool selection language model.

Since the primary objective in this step is to select the correct tool (but not execute the tool call), User: While I was coming back home from the office, I saw a kid in Audi Q7. Where can I buy this car within 10 miles?

Tool Choices:
- CarLocator: Lists car dealers given price range.
- BankAccount: Creates a bank account in a bank.
- CarFinder: Finds dealers given car model and radius.
- CurrentWeatherCity: Current weather given city.
- …[*more choices*] …

Thought: Since I need to find the car within 10 miles, "Car Finder" tool seems to be the right choice here. I need to use this tool.

Act: CALLTOOL[CarFinder()]

Figure 2: Illustrative training example from our synthetically constructed tool selection dataset `ToolSelect`. Given a user instruction and a set of tools to choose from, the output consists of reasoning notes ("Thought") and the final tool selection ("Act").

I am confused to choose one of these two classes. Here are their names and descriptions:

a CarLocator - Lists car dealers given price range.
b CarFinder: Finds dealers given car model and radius.

A contrastive question is a question that upon asking would resolve such confusion. Generate a contrastive question that I can ask myself whose answer would help me make the right choice.

Verification Question: What is the primary purpose of the class I need? Is it to find a car dealership based on a specific car model and location (CarFinder), or is it to list car dealerships within a given price range (CarLocator)?

Figure 3: Verification method for tool selection: a contrastive question is generated that can then be answered to help discern among the top two predicted tools.

synthetically generated tools and their corresponding descriptions can easily be used in this setting, as we do not require their actual inner workings (in order to execute them). In our generated dataset, each training sample is thus composed of a user instruction, a candidate set of tools that includes the ground truth tool, and a reasoning note elucidating the correct choice of tool. An illustrative training sample is given in Figure 2.

2.1.1 Synthetic Tool Library Generation

Generation Procedure We generate a set of synthetic tools along with their corresponding descriptions, which are used to build the training examples. We start by first manually annotating a "seed set" of eight tools and their descriptions. Subsequently, we employ the Llama-65B (Touvron et al., 2023a) model to generate additional tools using few-shot prompting with the manually annotated tools (specified in Appendix A.4.1). This process then involves multiple iterations of prompting with different random seeds, where the tools generated in each iteration are integrated into the prompt for subsequent iterations to generate more diverse tools. Specifically, in each iteration, for every newly introduced tool, we identify the most similar tool in the prompt based on cosine similarity using RoBERTa sentence similarity (Reimers and Gurevych, 2019). We replace the most similar tool in the prompt with the new addition, ensuring a balanced diversity of tools in the prompt. Using this iterative approach, we generate a total of 60 tools.[2] It is noteworthy to highlight that this process yields a diverse set of tools from various domains including travel, banking, and calendar, with almost no manual effort.

Generating Challenging Tool Sets In generating these synthetic tools, we endeavor to have a tool set that is diverse, but also sufficiently challenging. An overly simplistic training set would contain only easy choices (e.g., a weather tool versus an email tool) and this would impede the model's ability to generalize to challenging instances during test time. To address this, we generate two *related tools* for each of our previously generated 60 tools. Related tools are defined as tools closely resembling a given tool but differing in either functionality or parameters. For instance, "*Bank account for a person name* " and "*Bank account for an account number*" are related tools. We use only the tool names, and not the descriptions, for generating related tools. After manually annotating related tools for our seed set of eight tools, we generate two related tools for each of the remaining tools with few-shot prompting with these examples, as indicated in Appendix A.4.3.

Finally, after manual inspection and curation, our dataset contains a total of 173 tools.

2.1.2 Generating Training Examples

Using the generated tool library, we can now generate training examples for our tool selection dataset.

[2]These tools were manually reviewed, and 7 duplicates were removed.

This requires generating inputs (instructions), curating candidate lists of tools, and generating outputs (reasoning notes that explain which tools should be selected, and actions to call those tools).

Generating Instructions We first manually annotate three instructions per tool for the seed set of eight tools. Using these examples, we generate three instructions per tool for all remaining tools by few-shot prompting Llama-2 70B.

Curating Candidate List of Tools For each generated instruction, a candidate list of tools is created by randomly selecting 7 tools and adding the original ground truth tool for which we generated the instruction. To introduce complexity, for a subset of the training set, we deliberately create challenging samples by restricting the candidate set to include only the ground truth tool and its related tools. This deliberate selection aims to increase the difficulty level, as distinguishing among these options is inherently more challenging than with randomly selected tools from the entire set.

Generating Target Outputs After generating the set of instructions along with their respective ground truth tool and a candidate list of tools, we create a reasoning note for each sample elucidating the rationale behind the selection of the ground truth tool, which becomes the target output for that training example (see Figure 2). Such reasoning notes have been observed to enhance reasoning abilities (Wei et al., 2022; Yao et al., 2022; Lanchantin et al., 2023). Reasoning note generation is accomplished by prompting Llama-2-Chat-70B with the instruction, list of tools, and the ground truth tool, and asking the model why the tool was chosen. The exact prompt used is provided in Appendix A.4.2.

Our final dataset, called `ToolSelect`, thus contains 555 samples for our 173 tools, of which 75 samples are *hard* examples, featuring candidate tool sets that contain only the ground truth tool and its related tools.[3] The average number of candidate tools per instruction is 7.34 with minimum and maximum number of candidate tools being 2 and 8. The average length of a reasoning note is 1054 characters.

The goal of this dataset is to enable generalization capabilities to a wide range of possible tools and tool libraries, and thus to demonstrate effectiveness across diverse scenarios. During training, the user instruction and tool list in each sample have masked labels, and hence, they do not contribute to the loss and are not learned.

2.2 Tool Selection Verification

Despite our model being fine-tuned on the above dataset, tool selection mistakes can still happen, particularly for related tools that are hard to differentiate. Crucially, we observe that those tool selection predictions typically appear as the top few predictions – but selection between them is challenging.

At inference time, we thus perform the following procedure. Given an instruction:

- First, we use the fine-tuned tool selection model to zero-shot select a tool.
- We then remove the initially selected tool from the candidate set of tools, and generate a second prediction.
- We construct a verification question to make a fine-grained decision between the model's top two selections.

We employ Llama-2-Chat-70B to generate a contrastive verification question, where the prompt asks the model to ask a question that emphasizes the distinctions between candidate tools given their names and descriptions (see Appendix A.4.4 for the exact prompt used and Figure 3 for an instantiation of it). Self-asking the model regarding its predictions has been noted to reduce hallucinations (Press et al., 2022; Dhuliawala et al., 2023), suggesting that posing such verification questions could assist the model in validating its predictions. Since only names and descriptions are used for generating contrastive questions, they can be generated offline and utilized as needed to make the method more efficient. The answers to these contrastive questions are obtained by further prompting Llama-2-Chat-70B, and these are appended to the context. Finally, we select the tool by using our fine-tuned Llama-2 70B model, with the top-two tools as candidates. As the verification answer to the question is in the context this can help it select the right tool.

2.3 Parameter Generation & Verification

Parameter Generation Following tool selection, we generate parameters for the selected tool through few-shot prompting with Llama-2 70B, utilizing demonstrations specific to the selected tool,

[3]The data was manually reviewed, and 56 noisy and duplicate samples were removed.

which are assumed to be provided. Note that we do not use our synthetic tool selection dataset for parameter generation since the dataset does not contain this subtask. This procedure is only done with real tools at inference time, without prior fine-tuning.

Parameter Verification The generated parameters are then subjected to verification before finalizing the set, resulting in the final tool call. To validate the generated parameters, we obtain a second set of parameter predictions. These can be acquired using sampling or an alternative model for diverse options; in our experiments, we employ ReAct-style prompting (Yao et al., 2022) with Llama-2 70B to obtain them. Then, for each individual parameter, we formulate a multiple-choice question to contrast the two predictions and further prime Llama-2-Chat-70B to make a definitive choice between them, providing the parameter description and user instruction as indicated in Appendix A.4.5. The final parameter predictions are then aggregated to construct the tool call by few-shot prompting Llama-2 70B as in Appendix A.4.7.

3 Experiments

In our experiments, we assess the effectiveness of our method using publicly available real-life tools.

3.1 Tasks

We evaluate our proposed method on four tool-calling tasks: Weather, Cat, Home and Booking from ToolBench (Xu et al., 2023b) that involve using the REST APIs. The Weather, Home, and Cat tasks each comprise 100 evaluation samples, while the Booking task contains 120 samples. Each task includes API documentation, parameter descriptions, user instructions, and the corresponding ground truth API call pairs.

For each task, there are multiple tools available, where the entire benchmark consists of a total of 17 tools. However, instead of evaluating each task individually, we make it more challenging by pooling together all available tools. In other words, for each user instruction, the model is provided a candidate list of 17 tools. The ToolBench benchmark with 17 tools presents an ideal balance between maximizing the number of tools that could be accommodated within the context window without requiring the use of a retriever. By eliminating the dependency on a retriever, we could independently evaluate the impact of self-verification on performance. We follow the evaluation protocol set by the benchmark and use success rate as the metric, where the success rate of a predicted tool call is 1 if its API response exactly matches the response from the ground truth API call.

3.2 Baselines

We conduct a comparison with various tool-augmented LLMs and prompting baselines using Llama-2 70B and Llama-2-Chat-70B. Specifically, for tool-augmented LLMs, we compare with ToolLLM 7B (Qin et al., 2023), NexusRaven-V2 13B[4], and Qwen1.5-Chat-72B (Bai et al., 2023)[5]. ToolLLM, NexusRaven-V2, and Qwen1.5 utilize API documentation to generate tool calls corresponding to a given instruction.

For prompting baselines, we try two distinct approaches: (1) Single-step, where the model is prompted directly for an API call with a single demonstration per tool; and (2) Two-step, where we decompose the process into tool selection and parameter generation, prompting the model individually for each step, as in TOOLVERIFIER.

The Single-step method uses 1-shot single demonstrations of each of the (17) tools to accommodate the prompt within the context size.

For the Two-step method, we consider two variants for the tool selection stage:

- *0-shot:* We use a 0-shot prompt that asks to select from the list of tools, without any demonstrations for tool selection. See A.4.6 for the exact prompt.
- *1-shot:* We show one demonstration per tool: a user instruction and corresponding tool name.

For parameter generation in the Two-Step method, we use three demonstrations for the selected tool.

3.3 TOOLVERIFIER Details and Ablations

Our model is denoted as TOOLVERIFIER. For tool selection it uses 0-shot prompting with Llama-2 70B fine-tuned on our synthetic `ToolSelect` dataset to select two tools and finalize one through our proposed contrastive-question-based tool verification. Subsequently, we generate two sets of parameters by employing standard few-shot and ReAct-style prompting Llama-2 70B with three demonstrations, and finalize the parameter set using our proposed parameter verification.

[4] `https://nexusflow.ai/blogs/ravenv2`

[5] We attempted comparing with ToolAlpaca (Tang et al., 2023), however, it led to context overflow.

Method	Weather	Booking	Home	Cat	Average
Tool-Augmented LLMs					
ToolLLM 7B	18.00	0.00	0.00	11.00	6.90
NexusRaven-V2 13B	55.00	27.50	43.00	82	50.71
Qwen1.5-Chat-72B	74.00	55.00	52.00	89	66.90
Prompting Baselines					
Single-Step Llama-2 70B (1-shot)	70.00	7.50	85.00	83.00	58.81
Two-Step Llama-2 70B (1-shot tool selection)	80.00	34.17	85.00	78.00	67.62
Two-Step Llama-2-Chat-70B (0-shot tool selection)	77.00	64.17	84.00	83.00	76.43
ToolVerifier (without verification)	76.00	82.50	85.00	82.00	81.43
ToolVerifier (tool selection verification only)	84.00	82.50	85.00	83.00	83.57
ToolVerifier (param selection verification only)	81.00	84.17	88.00	96.00	87.14
ToolVerifier (tool verification+param verification)	**90.00**	**84.17**	**88.00**	**97.00**	**89.52**

Table 1: **Tool call (tool selection + parameter generation) results**. We report percentage (%) success rate for each task. Our fine-tuned Llama-2 70B model ToolVerifier, even without verification, results in higher performance compared to the baselines. Our proposed verification mechanism further improves the success rate by 8 points – with both types of verification, for tool and parameter selection, each giving a separate boost in performance.

We additionally compare against ablated versions of our method: with tool selection verification only (but not parameter verification), with parameter selection verification only (but not tool verification), and without verification (in either stage).

3.4 Experimental Results

Tool Call (Selection + Parameters) The complete tool call performance results are presented in Table 1. Our approach, ToolVerifier, outperforms all baselines both on average and individually across all tasks. ToolVerifier outperforms all compared tool-augmented LLMs by a significant margin. Comparing ToolVerifier with Single-Step 1-shot highlights the challenges in generating complete tool calls at once, emphasizing the efficacy of the two-step decomposition. It also surpasses Single-Step 1-shot and Two-Step 1-shot tool baselines by a substantial margin of more than 50 points on the challenging Booking task.

A comparative analysis between ToolVerifier with and without parameter verification illustrates that parameter verification significantly enhances performance, showing improvements of up to 14 points in the Cat task and 6 points in the Weather task, leading to an average improvement of 6 points across all tasks. Similarly, the comparison between ToolVerifier with and without tool verification demonstrates that tool verification contributes significantly to the performance, such as up to 8 points in the Weather task. Notably, both types of verification help, each giving a separate boost, as shown by comparing the without verification results to tool selection verification only and tool+parameter verification. These results underscore the significance of verification in both steps for the tool call success.

Tool Selection Only We report the performance of tool selection (choosing the tool correctly, but without generating parameters) in Table 2. ToolVerifier outperforms all baselines on average and individually across the majority of tasks as well. ToolVerifier performs better than almost all compared tool-augmented LLMs, demonstrating its superior performance. While Qwen1.5 performs well at selecting the right tool, it struggles to generate parameters correctly. In contrast, our method's focus on parameter generation, facilitated by the two-step approach, yields improved overall performance as shown in Table 1. A comparative analysis between ToolVerifier with tool selection verification and without underscores the substantial enhancement in performance achieved through the verification process. Specifically, in tasks such as Weather and Home, we observe that the verification procedure not only improves performance in specific examples of lower baseline performance, but also does not adversely affect cases where verification may be unnecessary.

ToolVerifier (both with and without verification) shows that our zero-shot Llama-2 70B fine-tuned on our synthetically generated dataset performs better than other baselines, including a 0-shot Llama-2-Chat-70B, with an improvement of up to 6 points. The average number of candidate tools

Method	Weather	Booking	Home	Cat	Average
Tool-Augmented LLMs					
ToolLLM 7B	27.00	22.00	84.00	26.00	38.90
NexusRaven-V2 13B	84.00	93.33	**100.00**	**98.00**	93.81
Qwen1.5-Chat-72B	**93.00**	95.00	99.00	96.00	95.71
Prompting Baselines					
Single-Step Llama-2 70B (1-shot)	79.00	43.30	**100.00**	**98.00**	78.32
Two-Step Llama-2 70B (1-shot tool selection)	86.00	45.00	**100.00**	92.00	79.05
Two-Step Llama-2-Chat-70B (0-shot tool selection)	83.00	75.80	99.00	97.00	88.09
ToolVerifier (without verification)	82.00	**98.33**	**100.00**	96.00	94.28
ToolVerifier (tool selection verification)	91.00	**98.33**	**100.00**	97.00	**96.67**

Table 2: **Tool selection results**. We report accuracy in percentage (%) for each task. Our fine-tuned Llama-2 70B model ToolVerifier, even without verification, demonstrates superior performance compared to prompting-based baselines, with a higher average performance. Our proposed tool selection verification mechanism contributes another 2.5% improvement in accuracy on average.

per instruction in the generated training data for tool selection is 7.34 which is notably smaller than the 17 tools encountered during test time. This difference underscores the generalization capability of our method, demonstrating its effectiveness across diverse scenarios. The performance of 1-shot baselines reveals the difficulty in selecting the appropriate tool from an unseen set using prompting-based approaches. In contrast, fine-tuning the model on our synthetically generated dataset with examples of using a diverse set of tools significantly improves tool selection accuracy. Moreover, the verification procedure further improves tool selection performance by an additional 2.4 points on average.

4 Analysis

4.1 Self-verification improves tool-augmented LLMs

Our proposed self-verification method does not require any specific training process. To demonstrate its effectiveness on tool-augmented LLMs, we experiment with ToolLLM 7B, NexusRaven-V2 13B, and Qwen1.5-Chat-72B. We obtain two sets of predictions as in ToolVerifier, where the first predicted tool is removed from the set of tools to obtain the second prediction. After tool verification, we identify the final selected tool. We obtain two parameter predictions using two different sampling parameters while generation. We then perform parameter verification to finalize the parameters and construct the tool call. The complete tool call success rate comparison, with and without self-verification, is presented in Table 3. We observe a significant improvement in average performance: 6 points for ToolLLM-7B, 9 points for NexusRaven-V2 13B, and 4 points for Qwen1.5-Chat-72B. In certain tasks, such as the Weather task, the success rate of NexusRaven-V2 improved by 23 points through self-verification. This demonstrates that self-verification can be effectively applied to tool-augmented LLMs, enhancing their performance. The tool selection results are in Appendix A.1, where we also note significant improvements in performance post tool verification.

4.2 Verification Question Analysis

Qualitative Analysis Verification questions should ideally reference the distinguishing characteristics between two given tools in order to best help the model consider the differences between the two choices. This capability is particularly crucial for closely related tools. For instance, the tools "Forecast Air Pollution" and "Current Air Pollution" both provide air pollution data, but for future and current times, respectively. Verification question generation by Llama-2-Chat-70B identifies this nuanced difference and articulates it in the verification question: *Are you looking for data on the current air pollution levels in a specific location, or do you need to forecast the air pollution levels for a future date in that location?* Responses to such questions precisely address the identified distinction. An example response is: *"It appears that the user is looking for current air pollution data for a specific location with latitude -24.7 and longitude -57.3. Therefore, the answer is: A. Retrieve current air pollution data for a specific location."* Inserting this response

Method	Weather	Booking	Home	Cat	Average
ToolLLM 7B	18.00	0.00	0.00	11.00	6.90
ToolLLM 7B + Tool, Param Verification	23.00	7.50	9.00	15.00	13.33
NexusRaven-V2 13B	55.00	27.50	43.00	82.00	50.70
NexusRaven-V2 13B + Tool, Param Verification	78.00	34.17	46.00	84.00	59.29
Qwen1.5-Chat-72B	74.00	55.00	52.00	89.00	66.90
Qwen1.5-Chat-72B + Tool, Param Verification	76.00	57.50	59.00	91.00	70.24

Table 3: **Tool verification and parameter verification improve tool call success rate for tool-augmented LLMs**. We report percentage (%) success rate for each task. Our proposed verification mechanism significantly improves the success rate of all tool-augmented LLMs.

into the context improves tool selection accuracy, guiding the model towards the correct choice. For more distinct tools, the model captures higher level differences. For example, for "Forecast Air Pollution" and "Get favorite cat images", the generated question is: *Which aspect are you more interested in: predicting environmental air quality or exploring feline visuals?*

Significance of Contrastive Questions To demonstrate the significance of contrastive-question-based verification, we conduct an experiment by zero-shot prompting Llama-2-Chat-70B to choose one tool from the top-2 without employing a verification question. Instead, we present the names and descriptions of the top-2 tools and frame it as a multiple-choice question, asking Llama-2-Chat-70B to make a selection. We experiment on the Weather task and the accuracy of Llama-2-Chat-70B is 70% whereas the accuracy of contrastive question-based verification is 91 %. This significant enhancement over straightforward prompting illustrates effectiveness of contrastive questions.

Instruction-Conditioned Verification In our proposed approach we generate verification questions using solely the names and descriptions of the top-2 selected tools, see Figure 3. We compare this to conditioning on the user instruction as well, by adding it to the prompt. Conditioning on the instruction during verification still shows improvement over the no-verification baseline (89 versus 82), however, slightly decreases performance compared to the non-user-conditioned verification, dropping accuracy from 91 to 89, perhaps because the decision is biased to be more similar to the original top choice being verified, which was also based on the instruction. Note that, using only names and descriptions has the benefit that the questions can be precomputed and cached.

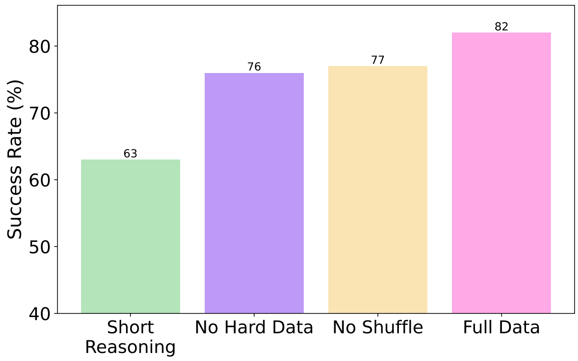

Figure 4: We analyze various aspects of our synthetic `ToolSelect` training data including the ordering of the candidate tool list (“No Shuffle”), difficulty level (”No Hard Data”), and the length of reasoning notes (“Short Reasoning”). We find samples with longer reasoning notes, difficult samples, and randomly ordered candidate tool lists contribute to high performance (“Full Data”).

4.3 Parameter Verification Error Analysis

In the parameter verification step, we identify a consistent pattern in errors while answering the verification questions, predominantly involving common sense errors where the model tends to hallucinate values instead of adhering to the user instruction, which is also observed in Mekala et al. (2023). A notable example of such errors occurs with the *min-price* parameter in Booking tool, which signifies the minimum price the user is willing to pay for a booking. In 5 instances out of 19 wrong predictions for the Booking task, when the user specifies only their maximum budget, the model generates the maximum value for the min-price parameter rather than 0. Similar errors are observed with the *min-area* parameter in the Home task. In 4 instances out of 12 mistakes, when the user expresses the desire for a home given only a maximum area, the model incorrectly predicts the mentioned value as the minimum, instead of using 0.

4.4 Synthetic Training Data Analysis

We analyze our synthetic `ToolSelect` training data through various ablations, with results on tool selection for the Weather task given in Figure 4.

Challenging training samples (samples that have a candidate tool list containing related tools to the ground truth tool, see subsubsection 2.1.1) are found to improve generalization. To assess the impact of these challenging samples, we remove them and train a model solely with easier samples ("No Hard Data"). The results indicate a notable 6-point drop in performance after excluding the hard samples, highlighting their significance.

Next, we experiment by reducing the maximum reasoning note length from 480 tokens to 200 tokens ("Short Reasoning") and observe a significant drop in performance, up to 19 points. Shorter reasoning texts are significantly less helpful in guiding appropriate tool selection.

Lastly, we compare performance with different orderings of the candidate tool list. In the "No Shuffle" scenario, the ground truth tool in the training data is always positioned first. Implementing this ordering strategy results in a 5-point drop in performance, underscoring the significance of randomly shuffling the candidate tool list in the training data.

More studies regarding parameter generation-only performance, and prompts are detailed in Appendix A.2, A.4 respectively.

5 Related Work

Self-Verification Iterative improvement of LLMs typically involves prompting an LLM to provide feedback on given generated facts or answers and subsequently refining their outputs (Madaan et al., 2023; Shridhar et al., 2023a; Lu et al., 2023) which has also been shown to reduce hallucination (Dhuliawala et al., 2023). Additionally, some studies involve the fine-tuning of custom LLMs to better accommodate feedback (Yu et al., 2023; Shridhar et al., 2023b; Zhang et al., 2023), aiming to enhance reasoning in chain-of-thought prompting for improved downstream performance. In this paper, we focus on tool usage, whereas previous works typically focus on generation. Our approach contrasts the choice between selecting options, whereas previous work typically verifies single facts or answers in responses.

Enabling Tool Use in LLMs Many approaches have emerged for enabling tool usage in LLMs, involving techniques such as few-shot prompting with tool-use demonstrations across diverse tool categories, including APIs (Qin et al., 2023; Chen et al., 2023), machine learning models (Shen et al., 2023; Patil et al., 2023), and code interpreters (Gao et al., 2022; Chen et al., 2022). Additionally, several approaches advocate for fine-tuning LLMs on custom-generated datasets tailored for tool usage (Schick et al., 2023; Tang et al., 2023; Parisi et al., 2022; Xu et al., 2023b; Patil et al., 2023; Srinivasan et al., 2023; Yang et al., 2023). Recent works introduce tool documentation (Hsieh et al., 2023) and tool tokens (Hao et al., 2023) to facilitate tool usage. Despite the plethora of works focused on enabling tool usage in LLMs, to the best of our knowledge none has explored verification methods for this purpose. This paper aims to fill this gap by introducing multi-step contrastive verification.

LLMs for Data Generation LMs have been used for generating training data for various tasks including classification (Mekala et al., 2021, 2022), semantic similarity (Schick and Schütze, 2021), and instruction tuning (Wang et al., 2022; Honovich et al., 2022; Xu et al., 2023a; Taori et al., 2023). Several works (Tang et al., 2023; Qin et al., 2023; Tang et al., 2023; Schick et al., 2023; Patil et al., 2023; Srinivasan et al., 2023) have employed LLMs to generate synthetic tools or tool use data.

6 Conclusion

In this paper, we present a self-verification method for enhancing the performance of tool calls for LLMs. This involves decomposing the tool call generation task into tool selection and parameter generation, where we apply verification at each step. Additionally, we open-source a synthetic dataset for improved reasoning and generalization to unseen tools. Experimental results on four tasks from the ToolBench benchmark demonstrate substantial improvements using our approach.

7 Limitations

Our self-generated verification questions and answers are produced in a zero-shot manner, making them effective for general-purpose tools but may necessitate further training for niche tools. Additionally, our framework is currently designed for single-tool-usage tasks and does not support instructions requiring multiple or compositional tool usage.

8 Ethics Statement

This paper introduces a self-verification method for tool calling that generates verification questions to aid in making accurate choices with confidence. As such, we do not expect that the fine-tuning self-verification process should introduce biases not already observed in the model, and we do not anticipate any significant additional ethical concerns beyond those issues already seen in standard systems (Weidinger et al., 2021).

9 Acknowledgments

The authors thank Omer Levy and Chunting Zhou for helpful discussions.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *ArXiv*, abs/2211.12588.

Z. Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji rong Wen. 2023. Chatcot: Tool-augmented chain-of-thought reasoning on chat-based large language models. *ArXiv*, abs/2305.14323.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-verification reduces hallucination in large language models. *ArXiv*, abs/2309.11495.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *ArXiv*, abs/2305.11554.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. *ArXiv*, abs/2212.09689.

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. *arXiv preprint arXiv:2308.00675*.

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. 2023. Learning to reason and memorize with self-notes. *arXiv preprint arXiv:2305.00833*.

Jianqiao Lu, Wanjun Zhong, Wenyong Huang, Yufei Wang, Fei Mi, Baojun Wang, Weichao Wang, Lifeng Shang, and Qun Liu. 2023. Self: Language-driven self-evolution for large language model. *ArXiv*, abs/2310.00533.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Dheeraj Mekala, Varun Gangal, and Jingbo Shang. 2021. Coarse2Fine: Fine-grained text classification on coarsely-grained annotated data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 583–594, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dheeraj Mekala, Tu Vu, Timo Schick, and Jingbo Shang. 2022. Leveraging QA datasets to improve generative data augmentation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9737–9750, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Dheeraj Mekala, Jason Wolfe, and Subhro Roy. 2023. ZEROTOP: Zero-shot task-oriented semantic parsing using large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5792–5799, Singapore. Association for Computational Linguistics.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *ArXiv*, abs/2205.12255.

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

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Timo Schick and Hinrich Schütze. 2021. Generating datasets with pretrained language models. *ArXiv*, abs/2104.07540.

Yongliang Shen, Kaitao Song, Xu Tan, Dong Sheng Li, Weiming Lu, and Yue Ting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *ArXiv*, abs/2303.17580.

Kumar Shridhar, Harsh Jhamtani, Hao Fang, Benjamin Van Durme, Jason Eisner, and Patrick Xia. 2023a. Screws: A modular framework for reasoning with revisions. *ArXiv*, abs/2309.13075.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ramakanth Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. 2023b. The art of llm refinement: Ask, refine, and trust. *ArXiv*, abs/2311.07961.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*.

Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Hanzi Mao, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

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*, abs/2306.05301.

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`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971.

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

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. In *Annual Meeting of the Association for Computational Linguistics*.

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*, 35:24824–24837.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. 2021. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*.

Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. 2023a. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. *ArXiv*, abs/2304.01196.

Qiantong Xu, Fenglu Hong, B. Li, Changran Hu, Zhe Chen, and Jian Zhang. 2023b. On the tool manipulation capability of open-source large language models. *ArXiv*, abs/2305.16504.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023c. Toolbench leaderboard. `https://huggingface.co/spaces/qiantong-xu/toolbench-leaderboard`. Accessed: Feb 15 2024.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *ArXiv*, abs/2305.18752.

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*.

Xiao Yu, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2023. Teaching language models to self-improve through interactive demonstrations. *ArXiv*, abs/2310.13522.

Haodi Zhang, Min Cai, Xinhe Zhang, Chen Jason Zhang, Rui Mao, and Kaishun Wu. 2023. Self-convinced prompting: Few-shot question answering with repeated introspection. *ArXiv*, abs/2310.05035.

A Appendix

A.1 Self-verification improves tool selection of Tool-Augmented LLMs

We apply our proposed self-verification on tool-augmented LLMs, and present their performance on tool selection alone in Table 4. We note significant improvements in tool selection accuracy, post tool verification. For instance, the average accuracy of ToolLLM 7B increases by 9 points, NexusRaven-V2 13B by 5 points, and Qwen1.5-Chat-72B by 4 points. This demonstrates that the tool verification enhances the performance of tool-augmented LLMs.

A.2 Parameter Generation Only Comparison

We additionally compare TOOLVERIFIER in the tool selection upperbound scenario, where the groundtruth tool selection is provided, and a model is only required to generate parameters through three-shot prompting. Results are given in Table 5. TOOLVERIFIER outperforms Llama-2-Chat-70B by 16 points as well as both Llama-2 70B and GPT-3.5-Turbo by an average of 6 points on a majority of the tasks, with an improvement of up to 14 points compared to Llama-2 70B in the Cat task and 8 points in the Home task compared to GPT-3.5-Turbo. TOOLVERIFIER also demonstrates superior performance compared to GPT-4 on Weather and Cat tasks by 6 and 4 points, respectively. This shows that our proposed method outperforms few-shot prompting approaches, even compared to stronger base models.

A.3 Self-Verification vs Self-Consistency

Our self-verification approach entails three separate LLM inferences: one for each model, Llama-2 70B and Llama-2-Chat-70B, followed by another to answer the verification question. To evaluate the performances of self-consistency and self-verification, we use the same number of inference calls for both methods and present a comparison in Table 6. The results indicate that self-verification achieves significantly greater improvements than self-consistency.

A.4 Prompts & Configurations

We use top-p sampling while generating with a temperature set to 0.7.

A.4.1 Tool Generation

The prompt for tool generation using few-shot prompting LLaMa-65B is:

```
Name: Humidity
Description: Computes humidity at a location on a date

Name: Trip Booking
Description: Makes a travel booking

Name: Currency Conversion
Description: Converts an amount from one currency to another.

Name: Age Calculator
Description: Calculates the age based on a given birthdate and the current date.

Name: Search Engine
Description: Searches online about a query

Name: Restaurant Finder
Description: The Restaurant Finder tool finds the restaurants based on its location, cuisine and the number of people.

Name: Movie Review
Description: The Movie Review tool gets top-rated movie reviews for a particular movie.

Name: Pizza Order
Description: The Pizza Order tool orders a pizza with provided toppings and size.

Name:
```

A.4.2 Reasoning Note Generation

The prompt for reasoning note generation using Llama-2-Chat-70B is:

```
[INST] «SYS»
You are a helpful assistant.
«/SYS»

Here are the list of available tools:
{Candidate tool list}

A user said, "{instruction}".

To answer this, you found Tool "{name}" to be the most suitable than other tools. Why?[/INST]
```

In the above prompt "{instruction}" denotes the user instruction and "{name}" denotes the ground truth tool. "{Candidate tool list}" contains names and descriptions of each tool.

A.4.3 Related Tools Generation

The prompt for related tool generation using few-shot prompted Llama-2 70B is:

Method	Weather	Booking	Home	Cat	Average
ToolLLM 7B	27	22	84	26	38.90
ToolLLM 7B + Tool Verification	34	30	95	33	47.14
NexusRaven-V2 13B	84	93.33	100	98	93.81
NexusRaven-V2 13B + Tool Verification	93	100	100	99	98.10
Qwen1.5-Chat-72B	93	95	99	96	95.71
Qwen1.5-Chat-72B + Tool Verification	97	100	100	99	99.05

Table 4: **Tool verification improves tool-augmented LLMs on tool selection**. We report accuracy in percentage (%) for each task. The tool verification improves ToolLLM 7B by 8 points, NexusRaven-V2 13B by 5 points, and Qwen1.5-Chat-72B by 4 points on average respectively.

Method	Weather	Booking	Home	Cat	Average
GPT-4*	93	**96.70**	**97**	96	**95.72**
GPT-3.5-Turbo*	90	85.80	80	92	86.90
Llama-2 70B	93	84.17	85	86	86.91
Llama-2-Chat-70B	89	45	91	88	76.67
ToolVerifier	**99**	85.80	88	**100**	92.85

Table 5: **Parameter generation results.** We report success rates (%) in the upperbound setting where the model is provided the ground truth tool selection, and must only generate parameters. We observe our fine-tuned Llama-2 70B model ToolVerifier outperforms Llama-2 70B and GPT-3.5-Turbo models in the majority of tasks and on average in this setting. Results with $*$ are taken from the Toolbench Leaderboard (Xu et al., 2023c,b).

Name1: Humidity
Name2: Humidity at timezone
Name3: Humidity Altitude Location date

Name1: Book Review
Name2: Book Review By Date
Name3: Book Review By Day

Name1: Car Rental
Name2: Car Rental with insurance
Name3: Car Rental with driver

Name1: {name}
Name2:

In the above prompt {name} denotes the name of the tool whose related tools are being generated. While generating multiple related tools per original tool, we generate one related tool after another with different seeds, to improve the diversity of the related tools.

A.4.4 Contrastive Question Generation

[INST] «SYS»
You are a helpful assistant.
«/SYS»

I am confused to choose one of these two classes. Here are their names and descriptions:
a. {name1} - {description1}
b. {name2} - {description2}

A contrastive question is a question that upon asking would resolve such confusion. Generate a contrastive question that I can ask myself whose answer would help me make the right choice.[/INST]

In the above prompt "{name1}", "{description1}" and "{name2}", "{description2}" are names and descriptions of two selected tools respectively.

Method	Weather	Booking	Home	Cat	Average
Llama-2 70B + Self-Consistency@3	84.00	80.83	85.00	83.00	83.09
ToolVerifier (tool verification+param verification)	**90.00**	**84.17**	**88.00**	**97.00**	**89.52**

Table 6: **Self-consistency vs Self-verification** We consider the same model and same number of inference calls (i.e. 3), and compare self-consistency and self-verificaiton. We report the percentage (%) success rate for each task. We notice that self-verification performs significantly better than self-consistency.

A.4.5 Parameter Verification

```
[INST] «SYS»
You are a helpful assistant.
«/SYS»

A user said, "{instruction}"

parameter definition

For the above user instruction, I am confused about choosing one of these two for "{parameter name}".
a. {prediction 1}
b. {prediction 2}

What is the answer? Answer the following question strictly based on what the user said above. If there is no mention, respond with "None". If there is, select the answer from the given options and respond with the chosen option only in square brackets []. [/INST]
```

In the above prompt "{instruction}" denotes the user instruction. "{parameter name}" represents the parameter name under verification. Additionally, "{prediction 1}", "{prediction 2}" signify two parameter predictions obtained from Llama-2 70B and Llama-2-Chat-70B, respectively.

A.4.6 0-shot Chat LLaMa-70B

```
[INST] «SYS»
You are a helpful assistant.
«/SYS»

Here are the list of available tools:

{Candidate tool list}

A user said, "{instruction}"

What tool to use for the above instruction? Respond with just the name of the tool[/INST]
```

In the above prompt "{instruction}" denotes the user instruction and "{Candidate tool list}" contains names and descriptions of each tool.

A.4.7 Tool Call Construction

```
INS: A user says, "Please retrieve the temperature, humidity, wind, and visibility data at place with latitude = -37.3, longitude = 1.9."
lat: -37.3
lon: 1.9
units: none
mode: none
lang: none
API: curl -X GET 'https://api.openweathermap.org/data/2.5/weather?lat=-37.3&lon=1.9&appid=API_KEY&units=none&mode=none&lang=none'

INS: A user says, "How is the weather now in location with longitude 125.9 and latitude 39.0? Respond in simplified Chinese with json format and imperial units."
lat: 39.0
lon: 125.9
units: imperial
mode: json
lang: zh_cn
API: curl -X GET 'https://api.openweathermap.org/data/2.5/weather?lat=39.0&lon=125.9&appid=API_KEY&units=imperial&mode=json&lang=zh_cn'

INS: A user says, "Give me a current weather report for place where longitude is 174.4 and latitude is -19.0."
lat: -19.0
lon: 174.4
units: none
mode: none
lang: none
API: curl -X GET 'https://api.openweathermap.org/data/2.5/weather?lat=-19.0&lon=174.4&appid=API_KEY&units=none&mode=none&lang=none'

INS: A user says, "{instruction}"
{param_str}
API:
```

In the above prompt "{instruction}" denotes the user instruction and "{param_str}" contains parameters and their predicted values.

A.4.8 Significance of Contrastive Questions

An example prompt is provided below.

[INST] «SYS»
You are a helpful assistant.
«/SYS»

A user says, "Please retrieve the temperature, humidity, wind, and visibility data for next week with latitude = -37.3, longitute = 1.9."

To address the above instruction which one of the below tools is the most suitable? Select the answer from the given options and respond with the chosen option ONLY in square brackets [].

A. Forecast Air Pollution = Get the future air pollution data in location with latitude={lat}, longitude={lon}
B. Forecast Weather Latitude Longitude = Get the weather data for future in location with latitude={lat}, longitude={lon}[/INST]

A.5 Hyperparameters for Llama-2 70B Fine-tuning

We fine-tune Llama-2 70B for 3 epochs with a learning rate of 1e-5 with warm up. The effective batch size is 8 and the weight decay is 0.1. We train it on 16 A100 GPUs.

A.6 Frequently Asked Questions

Why did you use LLaMa-65B for tool generation instead of Llama-2 70B? The 70B model was not released by the time we generated tools. Hence, we used the available 65B model.

Why only 4 tasks were chosen from ToolBench benchmark? Our framework is currently designed for single-tool-usage tasks. Therefore, we experiment on all single-turn tool calling tasks in the ToolBench dataset that can be completed with a single tool call without requiring additional actions and skills. In contrast, VirtualHome involves generating and executing a sequence of actions in a single step, which is outside our current scope. The Google Sheets task requires additional python coding skill, making it an unsuitable fit for our experiments. As a result, we have excluded these two tasks from our evaluation.

Why is it not evaluated on a benchmark with thousands of tools? To accurately assess the efficacy of our proposed self-verification and fine-tuning approach, we evaluate our methods on a benchmark that would not require any retrieval. By eliminating the dependency on a retriever, we could isolate the impact of our techniques and demonstrate a clear performance improvement. The ToolBench benchmark, comprising 17 diverse tools, presented an ideal balance between maximizing the number of tools that could be accommodated within the context window without requiring the use of a retriever. To make our method work for thousands of tools a standard approach would be to combine it with a retrieval system and then our method to do the final selection step from the top retrieved tools, which is an experiment that is beyond the scope of the paper.

The proposed framework prompts the model to select tools (or parameters) from two candidates. Why the candidate number is set to 2? We observed that the ground truth tool was typically in the top two tool selections. Further, asking contrastive verification questions is most natural/makes most sense as a comparison between two choices. Therefore, we choose two candidates. This gives a significant performance improvement.

B API Details

The four APIs pertaining to ToolBench are the Weather, Booking, Home, and Cat APIs. To execute the API calls, we registered for access to the Weather and Cat API, whereas for Home and Booking we ensured correct syntax, as proposed in the benchmark (Xu et al., 2023c).