On the Tool Manipulation Capability of Open-source Large Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent studies on software tool manipulation with large language models (LLMs) mostly rely on closed model APIs. The industrial adoption of these models is substantially constrained due to the security and robustness risks in exposing information to closed LLM API services. In this paper, we ask *can we enhance open-source LLMs to be competitive to leading closed LLM APIs in tool manipulation, with practical amount of human supervision*. By analyzing common tool manipulation failures, we first demonstrate that open-source LLMs may require training with usage examples, in-context demonstration and generation style regulation to resolve failures. These insights motivate us to revisit classical methods in LLM literature, and demonstrate that we can adapt them as model alignment with programmatic data generation, system prompts and in-context demonstration retrievers to enhance open-source LLMs for tool manipulation. To evaluate these techniques, we create *ToolBench*[1], a tool manipulation benchmark consisting of diverse software tools for real-world tasks. We demonstrate that our techniques can boost leading open-source LLMs by up to 90% success rate, showing capabilities competitive to OpenAI GPT-4 in 4 out of 8 ToolBench tasks. We show that such enhancement typically requires about one developer day to curate data for each tool, rendering a recipe with practical amount of human supervision.

1 Introduction

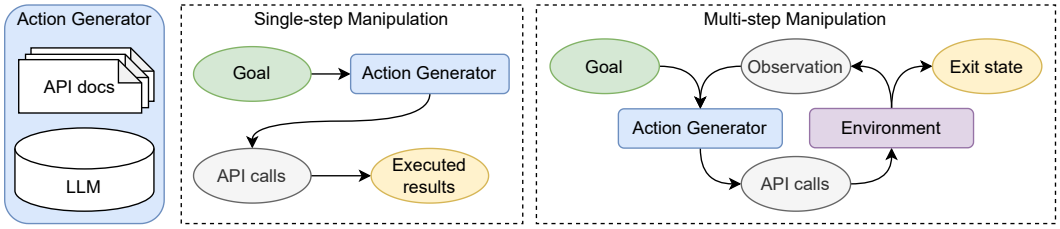
Tool-augmented large language models (LLMs) recently emerge as a research frontier. Such augmented LLMs demonstrate tool manipulation capabilities which automate software operations through natural language instructions (1; 2; 3; 4; 5). Despite the fact that open-source LLMs substantially shrink the quality gap towards proprietary closed LLMs in tasks such as chatbot (6; 7; 8; 9), recent tool-augmented LLMs still mostly rely on closed LLM APIs (1; 2; 3; 4). This leads to a fundamental barrier for the industrial adoption of these augmented LLMs due to security and robustness risks associated with exposing enterprise-internal workflows and information to closed LLM APIs (10; 11). To maximize the industrial impact, there is a substantial need for tool manipulation capabilities founded on open-source LLMs. To this end, we ask *can we build on open-source LLMs with practical amount of human supervision and achieve tool manipulation capabilities competitive to closed LLMs.*

In this paper, we first demystify key challenges for tool manipulation using open-source LLMs; we then leverage the insights to suggest practical recipes for enhancement. Concretely, we study the setting shown in Figure 1 where LLMs take in a natural language instruction as the goal and generate API calls to accomplish the goal. Although we expect a quality gap between the open-source and closed LLMs (12), what we observe is a far more severe disparity. Specifically, for an on-sale house searching tool, a leading open LLM for code generation fails every test case while the OpenAI GPT-

[1]Available at `https://github.com/sambanova/toolbench`

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Figure 1: Tool manipulation setup. We augment LLMs as action generators with access to API documentations. In a single-step scenario, an action generator directly generates API calls to accomplish the goal. A multi-step action generator iterates with an environment using API calls and generates the next-step calls based on the information from the environment until an exit state.



4 (13) attains 77% success rate across the same one hundred examples. This observation motivates us to study the challenges for open-source LLMs to attain strong tool manipulation capability.

During our investigation, we identify three key challenges that impede the performance of open-source LLMs in tool manipulation. Firstly, open-source models often struggle to accurately identify API names, whereas closed LLMs demonstrate the capability to invoke the correct APIs without explicit usage examples or documentation during inference. This suggests that closed LLMs hypothetically internalize knowledge of API usage during training. Secondly, we show that without demonstration examples, open-source LLMs often fail to populate the appropriate values for API arguments. Thirdly, we demonstrate that open-source LLMs tend to produce non-executable generation, such as natural language beyond the desired code.

Our insights suggest us to revisit three *simple* techniques from LLMs for conventional NLP tasks. In the context of tool manipulation, we adapt them with practical amount of supervision and use them to enhance open-source LLMs. *Model alignment:* To first internalize API usage knowledge, we perform instruction tuning (14; 15) with programatically generated data. Specifically, we first write a few dozens of templates on goals and corresponding API calls. We then pragmatically bootstrap the data volume by instantiating templates with concrete key word values. *In-context demonstration retriever:* Inspired by retrieval-augmented generation (16; 17; 18), we additionally enhance the LLMs with a *retriever* to leverage in-context demonstrations during inference. This module selects demonstration examples with the most semantically similar goals from a human-curated pool of examples. Given n API functions, the retriever only requires $\mathcal{O}(n)$ examples where every API function appears in at least one example. We then leverage LLMs to generalize to goals achieved by unseen API combinations. *System prompt:* Finally we embed goal descriptions into a pre-defined system prompt which provides inference-time guidelines to generate executable API calls; such system prompts were shown to regulate language style in chatbots (19). These techniques only require a small amount of human supervision. Thus they render a potentially practical recipe for building on top of open-source LLMs.

To extensively evaluate the inspired techniques, we present *ToolBench*, a benchmark suite on eight diverse tools ranging from Google Sheets manipulation to controlling robots (20). It enables the first publicly-available quantitative evaluation test bench among the ones brought up in the tool-augmented LLM literature (2; 3). For the software tools in our benchmark, LLMs need to accomplish a variety of goals by selecting and combining API functions from up to a hundred candidates.

Using the tools in the ToolBench suite, we first empirically show that leading open-source LLMs can demonstrate up to 78% lower success rate when compared to the OpenAI GPT-4 APIs. We then demonstrate that these simple techniques can substantially improve the success rate of open-source LLMs by up to 90%, attaining results competitive or better than OpenAI GPT-4 models in 4 out of the 8 tools in our benchmark[2]. To reveal the impact of different techniques, we provide evidence that aligning model with synthetic data primarily contributes to the significant improvement of open-source LLMs. The system prompt and the in-context demonstration retriever further enhance the performance. During the enhancement process, we observe that, on average, it takes just one day for a developer to craft the in-context demonstrations and curate the templates for generating model alignment data. This implies that the recipe requires a practical level of human supervision.

[2]We apply the same system prompt and in-context example retriever for GPT-4. Model alignment is not applicable to GPT-4 as there is no publicly available tuning APIs for it during our experiments.

Our contributions and the structure of this paper are as follows.

- In Section 3, we reveal challenges in API selection, argument populating and non-executable generation which hinder open-source LLMs on tool manipulation.
- To alleviate the challenges, we revisit simple techniques for conventional NLP tasks. We adapt them for tool manipulation to boost open-source LLMs with minimal human supervision in Section 4.
- In Section 5, we introduce the ToolBench, the first open-sourced benchmark with pre-defined test cases for quantitative evaluation compared to the ones in the recent tool-augmented LLM literature.
- We demonstrate in Section 6 that our adapted techniques boost open-source LLMs by up to 90% success rate, showing competitiveness with GPT-4 APIs in 4 out of 8 ToolBench tasks.

2 Background

To establish backgrounds, we first concretize the software tool manipulation setup. We then present a preliminary observation on the capability of open-source LLMs. This observation motivates our study on the challenges in Section 3 which inspire simple techniques for enhancements in Section 4.

2.1 Tool manipulation setup

In this paper, we study the scenario where software users intend to translate a natural language goal description g into a sequence of application programming interface (API) calls $C_g = \left\{c_0, c_1, \cdots, c_{n_g}\right\}$ to accomplish the goal. We study tool manipulation with open-source LLMs in this specific setting, because APIs serve as the prevalent abstraction for developers and users in modern software systems.

Large language model Autoregressive language models encode probabilities of the next word x_{N+1} given $x_0, x_1, \cdots, x_N$ as the context sequence (21). By sampling from this conditional probability $p\left(x_{N+1}|x_0, x_1, \cdots, x_N\right)$ iteratively, it generates language continuations from given contexts. In the recent wave of scaling up model size and training data volume, transformer-based language models show unprecedented capability in instruction following for text and code generation (22; 23; 24). In the context of tool manipulation, we cast goal descriptions and optional information as an instruction in the context and task the LLMs to generate code for API calls as the continuation.

Action generator A key implementation for tool manipulation is an action generator $\mathcal{A}$ which maps a goal g to API calls C_g. As open-source LLMs likely have not seen the information regarding the relevant APIs, we augment an LLM $\mathcal{M}$ into an action generator by providing access to a pool of m candidate API functions $\mathcal{D} = \{d_0, d_1, \cdots, d_m\}$. Due to the input sequence length limit of LLMs, we provide an optional retriever $\mathcal{R}$ to retain a relevant subset of API documents $\mathcal{D}_g = \mathcal{R}\left(g, \mathcal{D}\right) \in \mathcal{D}$. Thus, the action generator produces the sequence of API calls $C_g = \mathcal{A}\left(g, \mathcal{D}_g, O\right)$, where O represents the optional information that can be included in the prompt. This is a naive way of retrieval augmented generation (18; 25; 26) and we employ an off-the-shelf retriever implementation (27) for our study, but we also highly encourage the community to explore algorithms tailored for the action generator.

Algorithm 1 API Call Generation

Input: Goal g, API docs $\mathcal{D}$, action generator $\mathcal{A}$
Input: Optional info O
procedure ACTIONGEN(g, $\mathcal{D}$, $\mathcal{A}$, O)
  $\mathcal{D}_g \leftarrow \mathcal{R}\left(g, \mathcal{D}\right)$ ▷ Retrieve API functions
  $C_g \leftarrow \mathcal{A}\left(g, \mathcal{D}_g, O\right)$ ▷ API call generation
  return C_g
end procedure

Single and multi-step tool manipulation As shown in Figure 1, an action generator may interact with software in either a single-step or a multi-step scenario. In a single-step scenario, action generator directly produces an API call sequence $C_g = \mathcal{A}\left(g, \mathcal{D}_g, \emptyset\right)$. In a multi-step scenario, the action generator produces a series of API call sequences $C_g = \cup_i C_{g,i}$ where each segment $C_{g,i}$ is used to interact with a predefined environment $\mathcal{E}$ and generates the observation $O_i = \mathcal{E}(C_{g,i})$. The observation is then used to generate a new segment $C_{g,i+1} = \mathcal{A}\left(g, \mathcal{D}_g, O_i\right)$. The process stops at an exit state. Throughout the remainder of this paper, we use the single-step setup for illustration clarity unless stated otherwise. Our experiments in Section 6 cover both single and multi-step cases.

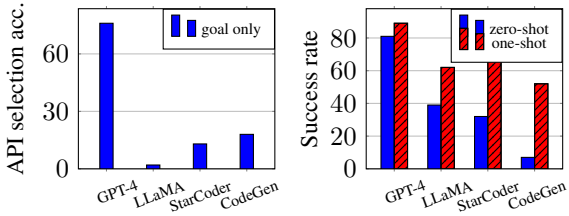
2.2 Motivating Observation

Table 2: Example of tool manipulation errors. Errors are highlighted in red.

Goal	# To move the robot to position (x, y)
	robot.move_to(x, y)
	# To raise the arm by a given height
	robot.raise_arm(height)
	Task: how to move a robot to (20, 30)?

Expected results	robot.move_to(20, 30)
Wrong API	robot.**raise_arm**(20)
Wrong Arguments	robot.move_to(**30, 20**)
Non-executable	**You can create a robot with** **robot = Robot()** **and move it to the target location by** robot.move_to(20, 30)

Figure 2: Without API documentation exposure during inference, closed LLMs attain high accuracy in selecting APIs (left), implying potential example usage exposure during training. Hand-picked oracle one-shot demonstration improves success rate over zero-shot on the OpenWeather (right), showing the roofline impact of in-context demonstrations.



To assess the tool manipulation capability of open-source LLMs, we compare them to OpenAI GPT-4 API using the setup discussed in Section 2.1. In this preliminary comparison, we initially anticipate the closed LLMs go exhibit an advantage in tool manipulation, as observed in traditional NLP tasks (12). However we observe a significantly larger gap than expected. For instance, in a home search task, open-source LLMs have a hard time to generate correct API calls, resulting in a 70% success rate gap compared to the zero-shot GPT-4 APIs as shown in Table 1. Such gap motivates us to study what impedes open-source LLM' performance.

Table 1: Huge capability gaps on a house searching task. Open-source LLMs lag behind the OpenAI GPT-4 by 70% on success rate.

Model	GPT-4	LLaMA	StarCoder	CodeGen
Open source	✗	✓	✓	✓
Success rate	77%	0%	7%	0%

3 Challenges for open-source LLMs

To demystify key challenges, we study the behaviors of open-source LLMs in tool manipulation. By analyzing common mistakes in a weather query task, we discover three challenges to attain strong tool manipulation capabilities. As shown in Table 2, we observe that open-source LLMs often face difficulty in (1) API selection, (2) API argument population, and (3) generating legitimate and executable code [3]. These insights are described in detail in this section and inspire the techniques to alleviate the challenges in Section 4.

Table 3: Categorized typical tool manipulation error types on a weather query tool.

	GPT-4	LLaMA	StarCoder	CodeGen
Failure rate	19%	61%	68%	93%
API selection	0%	22%	22%	30%
Args. populating	14%	32%	23%	63%
Non-executable	5%	7%	23%	0%

Difficulty in API selection We observe that API selection failures often involve using incorrect APIs and even hallucinating non-existent API names. To quantitatively understand the intrinsic capability in API selection, we compare open-source LLMs to GPT-4 without providing any documentation or in-context demonstrations during inference. The results, as shown in Figure 2 for the weather query tool OpenWeather, reveal that GPT-4 can choose the right API without additional information beyond the goal, while open-source models struggle. Such capability disparity entails that *closed LLMs potentially internalize knowledge of API usage during training*.

Confusion in populating arguments After the action generator selects the appropriate APIs, the subsequent challenge lies in parsing the goal description and populating the API arguments. At this stage, we observe that open-source models often provide wrong values for the required API arguments. The confusion in argument populating contributes to up to 63% of the failures in open-source models,

[3] If a failure case has multiple errors, we categorize it by the first triggered category in the following order: non-executable generation, wrong API selection, wrong argument populating

as shown in Table 3. In an attempt to mitigate this issue, we provide the LLMs with a hand-picked oracle in-context demonstration which achieves the same goal with different argument values. We show in Figure 2 that the hand-picked oracle examples improve success rates by up to 45%. It is important to note that oracle examples are not intended as a solution for argument populating confusion, as they are hand-picked on a per-test-case basis. Nonetheless, these observations suggest that *in-context demonstrations can substantially enhance open-source LLMs for tool manipulation*.

Non-executable generation The third common failure of open-source LLMs is non-executable generation. Such failures encompass issues such as language verbosity around API calls and adherence to natural language based guidelines, as shown in Table 2. Open-source models sometimes exhibit such errors in 23% of one hundred weather query cases. These observations underscore *the necessity of regulating open-source LLMs to exclusively generate code.*

4 Boosting Open-source LLMs for Tool Manipulation

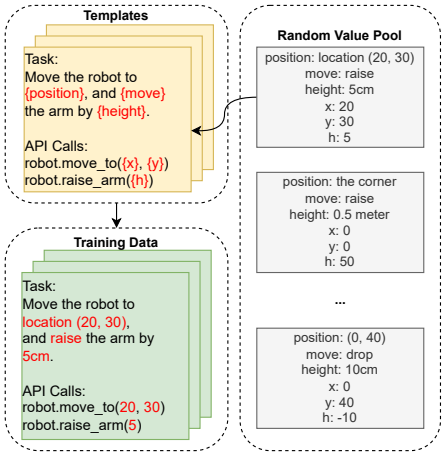
The insights from Section 3 emphasize the importance of tuning with API usage examples, in-context demonstration and generation regulation in the domain of tool manipulation. In this section, *we revisit three techniques from the LLM literature and adapt them to address the aforementioned challenges, using a practical amount of human supervision*. We first introduce model alignment with programatically curated data to internalize API usage knowledge in Section 4.1. We then discuss augmenting open-source LLMs with an in-context demonstration retriever in Section 4.2. Lastly, we apply a system prompt to regulate generation in Section 4.3. These techniques collectively serve as a strong baseline for alleviating the challenges presented in Section 3 and inspiring further innovations.

4.1 Multi-tool model alignment with programmatic data curation

Model alignment, through tuning LLMs with usage examples, plays a vital role in improving LLMs for capabilities such as instruction following and conversation (14; 19; 28). In light of our insights from in Section 3, we recognize the potential of model alignment with API usage examples to improve API selection capability. To practically leverage such alignment for tool manipulation, it requires a data curation strategy without massive manual example writing. Towards this end, we prototype a method which generates usage examples from human-curated templates.

Figure 3 depicts our flow to generate alignment data. We create a handful of templates consisting of goal descriptions and corresponding API calls. These templates contain one or more placeholder pairs. Each of these pairs maps to a key word in the goal and an argument in the corresponding API calls. We also provide a pool of candidate values for each keyword and randomly choose values to fill in the placeholders within the template. Given a tool with n candidate APIs, we only require $\mathcal{O}(n)$ human-curated templates to ensure practical human supervision. Specifically we use a principle where each of the n APIs is encouraged to appear in at least one template. In practice, we find it takes on average one day for one developer to curate the data for one software tool in our benchmark; this includes writing the goal templates, providing the pool of argument values and generate the data. We provide example templates we use for different tools in Appendix C. With data curated for all the tools, we perform model alignment tuning *jointly for all tools and produce a single model*.

Figure 3: Programmatic training data generation using templates and random values



4.2 Demonstration retrieval

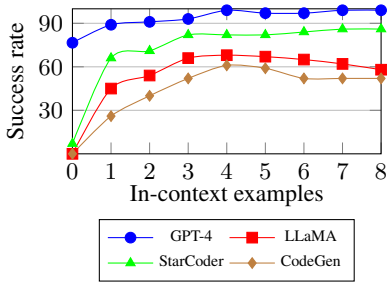
In Section 3, we demonstrate the efficacy of hand-picked oracle examples in improving argument populating. However, extending from oracles to practical in-context demonstration poses two challenges. First, given n API function candidates, there are exponentially many combinations of API calls associated with different goals. Thus, LLMs should be capable of generalizing to a wide

variety of goals based on a limited number of examples. Second, to ensure effective demonstration, it is important to provide LLMs with only the relevant examples without human interventions.

To fulfill the above two desiderata, we augment open-source LLMs with a demonstration retriever module. This module revolves around a repository where every API is required to appear in only one human-curated demonstration. This implies that only $\mathcal{O}(n)$ examples are needed. Among these demonstration examples, the retriever selects the most semantically similar examples to the goal descriptions.

Validation To verify the effectiveness of demonstration examples in practice, we empirically show that the retrieved demonstrations can improve the success rate on goals requiring API combinations unseen in the example repository. In particular, we evaluate this approach on the home search task which exposes 15 API functions and requires multiple functions to accomplish each goal. With only 10 human-curated demonstrations that do not precisely match any of the 100 test cases in terms of API combinations, the retrieved demonstrations can boost the success rate by up to 79% across open-source LLMs and make GPT-4 nearly perfect, as shown in Figure 4. This shows that the demonstration examples can improve tool manipulation for unseen types of goals with a repository of size $\mathcal{O}(n)$ only.

Figure 4: In-context demonstration can improve both closed and open-source models on Home Search, a tool for browsing houses on sale.



4.3 Generation regulation with system prompts

The use of system prompts is a well-established technique in chatbots powered by LLMs (19). By incorporating human-chatbot conversations, system prompts can effectively control the natural language style of the generated responses. In the context of tool manipulation, we regularize open-source LLMs to exclusively generate API calls with a system prompt in Figure 5, where the black part is the template shared across all tasks and the red rows are instantiated during inference for a certain goal. Our system prompt first defines a format that combines text sections containing goals, demonstrations, and generations. It then provides explicit guidelines in natural language, instructing the LLMs to generate code exclusively. The system prompt incorporates the goal description and the retrieved API functions directly for each request, reducing the human development effort to a one-time task.

Figure 5: System prompt with guidelines to only generate code in a desired format. Red parts are populated with real data for each test case during inference.

```
... As the AI API Assistant, my focus is on assembling the APIs in
the correct order to achieve the desired outcome without
introducing any new APIs or unnecessary information. ...

Here is a list of API functions:
{A list of API functions}
----------
To guide me in generating the executable code snippet, use the
following format. Within the three ticks I will be generate code only:
Task: Goal
Action:
```
API_function_name1(args1, args2)
API_function_name2(args3)
```

Here is a list of demonstration examples:
{A list of demonstration examples}
----------
Task: {Goal}
Action:
```
```

## 5 ToolBench: A New Tool Manipulation Benchmark

To evaluate open-source LLMs in the domain of tool manipulation, we curate a benchmark suite from both existing datasets and newly collected ones. This benchmark stands out as the first open-source test bench with predefined test cases for quantitative evaluation, distinguishing it from recent tool manipulation research using closed LLMs (2; 3). In this section, we introduce the software tools and the evaluation infrastructure. We also demonstrate the level of challenges posed by each tool, in terms of the ability to generalize to unseen API combinations and the requirement for advanced reasoning.

### 5.1 Software tools and evaluation infrastructure

As shown in Table 4, our benchmark consists of five tasks we collected and three tasks derived from existing datasets, including VirtualHome(29; 30), Webshop(31) and Tabletop(20). They cover both single-step and multiple-step action generation, which requires selecting and combining from 2 to 108
```

Table 4: Tasks in the ToolBench. We provide demonstration examples for few-shot in-context-learning while test cases are for quantitatively evaluation. We develop API complexity, a metric to quantify the challenge level in generalizing to unseen API combinations; higher complexity indicates more challenging tasks. We package the challenges beyond API complexity as advanced reasoning. We refer to Appendix A for more details on these tasks.

Task	Single Step						Multi-Step	
	Open Weather	The Cat API	Home Search	Trip Booking	Google Sheets	VirtualHome	WebShop Long / Short	Tabletop
Data								
API functions	9	6	15	20	108	40	2	32
Demonstration examples	18	12	10	11	10	83	1533 / 200	74
Test cases	100	100	100	120	70	100	100	105
Level of challenges								
API complexity	2.2	1.4	7.3	11.1	8.4	12.3	0.0	4.6
Advanced reasoning					✓		✓	✓

API functions to accomplish the goals. Each task consists of approximately approximately 100 test cases, including goal descriptions and the ground truth API calls. We also provide a limited number of demonstration examples to aid model predictions[4]. We include a comprehensive introduction and analysis of each task within the benchmark in Appendix A.

We use *success rate* as the primary evaluation metric for most tasks, except for the WebShop where we report rewards, as well as for VirtualHome where we use executability and Longest Common Subsequence (LCS), following the original metrics proposed by the respective authors. To facilitate evaluation, we build an infrastructure that executes the API calls generated by the action generators and assess the final outcome. This process enables reliable evaluation of tool manipulation capabilities without restricting the action generators to perfectly match the ground truth API calls.

5.2 Level of challenges

To assess the level of challenge, we examine ToolBench tasks based on their API complexity and the requirement for advanced reasoning. Intuitively, API complexity indicates the challenges in generalizing to unseen API combinations and non-default argument values. Challenges beyond API complexity then involve advanced reasoning.

Table 5: A typical task of Google Sheets manipulation. It requires both selecting the correct API function and reasoning on the arguments.

Product	Cost	Price
beef	1	3
pork	5	4
chicken	10	11

Task: Update beef's price to 10.
Action:
worksheet.update("C2", 10)

API Complexity To quantify the challenge in generalizing to unseen API combinations, we develop a task-agnostic complexity score $S \in \mathbb{R}_0^+$, where

$$S(\mathcal{T}, \mathcal{X}, \mathcal{D}) = \mathbb{E}_{t \in \mathcal{T}} \min_{e \in \mathcal{X}} d(t, e). \quad (1)$$

It averages over all the test samples in the test set $\mathcal{T}$ on the minimum distance between t and any demonstration example e from the example pool $\mathcal{X}$. In particular, the distance $d(t, e)$ between each test sample t and a demonstration example e is negatively proportional to the probability of transforming the API combination of e to match that of t, by randomly dropping the API functions irrelevant to t and inserting the uncovered API functions required by t from the API pool $\mathcal{D}$. We refer to the details of the complexity score to Appendix D and list their values in Table 4. The score is non-negative and the higher the score is, the more complex a task is. Despite the fact that this complexity score reflects the challenge level of API selection, it does not capture all the difficulties of a task. A task with low complexity score can still be very challenging as it might require advanced reasoning. For instance, even though Webshop is challenging, the API selection complexity of it is zero. This is because there are only two API functions requiring only one argument each in Webshop, and they are both covered by the examples, so there is no API selection complexity.

[4]For WebShop, we find that more than $\mathcal{O}(n)$ demonstration examples can improve the success rate. Nonetheless, these examples can be acquired from programmatic software operations without heavy human curation.

Advanced reasoning Within our benchmark, advanced reasoning encompasses challenges beyond generalizing to unseen API combinations. These challenges include non API-based coding for tasks such as Google Sheets and Tabletop, as well as decision-making based on observations returned from the WebShop environment. For instance, in the Google Sheets example shown in Table 5, the coordinate of the beef price's cell ("C2") cannot be easily derived from either the goal or the table itself. The action generator needs to understand the content or write additional python code to derive this coordinate before calling the API function. In the similar scenario, WebShop task requires the action generator to extract the exact button ID to click on the webpage given the description. These challenges, categorized as advanced reasoning, complement the API complexity category.

6 Experiment

In this section, we leverage the ToolBench to empirically validate the techniques introduced in Section 4. First, to concretize the capability gap between open-source and closed LLMs, we demonstrate that OpenAI GPT-4 API can have substantially higher success rate than representative open-source LLMs in Section 6.2. We then show in Section 6.3 that the simple techniques in Section 4 can boost open-source LLMs to achieve success rates competitive to in-context-learning with GPT-4 APIs[5] in four out of the eight tasks. Through ablation studies in Section 6.4, we additionally show that model alignment does the heavy lifting for boosting open-source LLMs, while system prompt and in-context learning robustify LLMs for further improvement.

6.1 Experiment Setup

To establish strong baselines, we use GPT-4 API as the representative closed LLM in our study because it attains the leading accuracy in mainstream NLP tasks. In our study, we compare LLAMA-30B (32), StarCoder (33) and CodeGen-16B-mono (34) to GPT-4. LLAMA represents open research models, while StarCoder and CodeGen are publicly available for both research and commercial purposes. We choose these three models due to their superior performance on ToolBench among open-source models as shown in Table 9[6]. In our experiments, we consider the zero-shot setting as the out-of-the-box configuration where only API documentation is provided without any demonstration examples. We use this configuration to understand the initial gap in capabilities among models. We then incorporate all available techniques on top of this initial configuration to assess their benefits. For the original Tabletop dataset (20), which includes examples in a few-shot setting without explicit API definitions, we only evaluate settings with in-context demonstrations. More detailed setup information is included in Appendix C. We run each job 3 times with different random seeds and report average accuracy. The variation is minimal, so we ignore them in the main paper but report them in appendix.

6.2 Capability Gap

Table 6 exhibits significant disparities in tool manipulation between the closed GPT-4 API and open-source models in the out-of-the-box zero-shot setting. For simpler tasks, namely Open Weather and the Cat API, which require only one API call for each goal, the open-source models exhibit success rates up to 74% lower than GPT-4. Furthermore, on all the remaining tasks other than the Webshop, none of the LLAMA, the StarCoder and the CodeGen model can reach meaningful accuracy or compare with GPT-4. These results highlight an opportunity to enhance open-source LLMs.

6.3 Boosting open-source LLMs

To boost the open-source LLMs, we first perform model alignment using programmatially generated data. We then apply a system prompt and a 3-shot demonstration retriever during inference. Given GPT-4 does not provide tuning APIs, we enhance the out-of-the-box GPT-4 with the same system prompt and demonstration retriever as the baseline. The improvements from the combined enhancement techniques are shown in Table 6, where the success rates of the open-source LLMs can improve up to 90%. As a result, the open-source models achieve competitive or better success rates on 4 out

[5]GPT-4 tuning APIs were not released by the time this work is done.

[6]Surprisingly, we observe that for tool manipulations, open-source LLMs instruction-tuned for conventional NLP tasks do not outperform their base models before tuning.

Table 6: Capability gap in tool manipulation is substantial between closed API and open-source LLMs in the out-of-the-box zero-shot setting. Using model alignment, the in-context demonstration retriever and the system prompt, open-soured LLMs attain significant boost in success rate. GPT-4 is enhanced with the retriever and system prompt. Tabletop is only evaluated in the few-shot fashion.

Task	Open Weather	The Cat API	Home Search	Trip Booking	Google Sheets	VirtualHome	WebShop Long	WebShop Short	Tabletop
Zero-shot Baseline									
GPT-4	81.3	97.4	76.6	91.5	5.7	40.8 / 8.0	0.0		-
LLaMA-30b	39.0	49.0	0.0	0.0	0.0	78.0 / 0.3	0.0		-
StarCoder	32.0	71.0	7.0	13.3	5.9	22.0 / 3.7	0.0		-
CodeGen-16B-mono	7.0	78.0	0.0	0.0	1.4	4.0/ 1.0	0.0		-
Enhanced w/ techniques									
GPT-4	99.0	98.0	98.0	99.2	68.6	29.0 / 21.7	0.0	0.0	83.8
LLaMA-30b	100.0	94.0	87.0	85.8	2.9	16.0 / 24.3	0.0	0.0	7.5
StarCoder	99.0	97.0	83.0	80.8	21.2	31.0 / 18.4	0.0	0.0	13.9
CodeGen-16B-mono	97.7	99.0	82.0	77.5	19.8	29.0 / 17.2	0.0	3.5	16.2

of 8 tasks, including Open Weather, the Cat API, VirturalHome and WebShop. Moreover, on Home Search and Trip Booking, the gap between the LLAMA model and the GPT-4 API is reduced to 11% and 13.4% respectively, compared to the initial gap of up to 91%. Despite the fact that open-source models are still lagging behind on the Google Sheets and Tabletop, these observations show that *our recipe can significantly improve the performance of open-source LLMs and attain success rates comparable to GPT-4 API on many of the ToolBench tasks*.

Human supervision To identify the practicality of an enhancement recipe, the amount of required human supervision is a crucial factor. In our approach, human supervision is primarily in the form of in-context demonstration examples and alignment data templates. Regarding the demonstration examples, we provide 10 to 83 examples for each task as shown in Table 4, except for WebShop given its difficulty in advanced reasoning. As shown in Table 10, the number of templates for alignment data is typically less than 100 for each task. We observe that providing these supervisions takes one developer day on average, making it practical in terms of the time cost on human supervision.

Remaining challenges In our experiments, we observe that the boosted open-source LLMs still have relatively low success rates on tasks that require advanced reasoning, such as Google Sheets, WebShop and Tabletop tasks. This implies the need to further enhance the reasoning capabilities of open-source models. We are excited about the prospect of more exploration from the community to address the challenges for tool manipulation on these complex tasks.

6.4 Ablation Study

We break down the contribution of the techniques in two ways. First, we apply each technique individually on top of the out-of-the-box zero-shot configuration and evaluate its impact. As shown in Table 7, both the 3-shot in-context demonstration and model alignment techniques bump up the success rates across all tasks, while the system prompt only benefits simple tasks that involve relatively fewer API calls for each goal.

Table 7: The number of ToolBench tasks improved (+N) or hurt (-N) over the baselines when adding or dropping techniques.

	LLaMA	StarCoder	CodeGen
Zero-shot	-	-	-
+ Sys. Prompt	+4	+4	+4
+ 3-shot	+8	+8	+8
+ Alignment	+7	+7	+7
Full system	-	-	-
- Sys. Prompt	-0	-2	-3
- 3-shot	-3	-4	-5
- Alignment	-5	-5	-7

Next, we consider the combination of all techniques and remove them one at a time to evaluate their relative contributions within the full system. As shown in in Table 7, solely removing model alignment triggers success rate degradation in up to 7 tasks, while removing either in-context demonstration up to 5 tasks and dropping system prompt up to 3. We notice that the tasks that are not significantly impacted when removing techniques are typically the ones with relatively low success rate (usually <20% even in the full system). Thus, those accuracy changes are hypothetically subject to high variance and fluctuation. The full results from the experiments in this section can be found in Table 12.

7 Related work

Our work establishes a strong connection to the LLM-driven program synthesis. In contrast to the conventional rule-based code generation in popular compilation frameworks (35), recent auto-regressive LLMs such as CodeGen(34), SantaCoder(36) and StarCoder(33) treat the problem as a sequence generation task and demonstrate superior capabilities in emitting semantically correct computer programs. We use CodeGen as a representative from these models in our study for API call generation.

Tool manipulation are also known as tool augmented learning (3; 37). Some of the works seek to augment generations with the execution results from various tools(1; 38; 39; 26; 40; 41; 42), while another line of works focus on executing the tools themselves, including embodied robotic learning (20; 30; 43; 44; 45), and automation for other tools (31; 46; 47; 48). We focus on the study of the second stream with different models and techniques.

Recent works in tool manipulation with LLMs mostly study techniques to enhance in-context-learning with closed LLMs APIs (1; 2; 3; 4; 5). In contrast, we study simple techniques to allow for developers to practically build on top of open-source LLMs. The three techniques we mention in this paper (19; 22; 26; 49) are well studied in the conventional NLP tasks. We revisit and adapt them in the context of tool manipulation on open-source models with a practical amount of human supervision. In the recent LLM literature, there are several works presenting tool manipulation benchmarks (2; 3). Compared to these benchmarks, the ToolBench is the first one providing predefined test cases for evaluation on real execution results.

8 Conclusion

In this paper, we answer the question *can we enhance open-source LLMs to compete with leading closed LLM APIs in tool manipulation, with practical amount of human supervision*. Drawing from our observations of the common tool manipulation failures and insights from the literature on conventional NLP tasks with LLM, we propose to instantiate model alignment with programmatical data generation, system prompts, and in-context demonstration retrievers to improve the tool manipulation capability of open-source models. To comprehensively evaluate the impact of these techniques, we create the *ToolBench*, a benchmark consisting of diverse software tools for real-world tasks. Our results demonstrate that these techniques can make the leading open-source LLMs competitive with the OpenAI GPT-4 in 4 out of 8 ToolBench tasks, all achieved with a practical amount of human labeling effort.

References

[1] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint arXiv:2302.04761*, 2023.

[2] M. Li, F. Song, B. Yu, H. Yu, Z. Li, F. Huang, and Y. Li, "Api-bank: A benchmark for tool-augmented llms," *arXiv preprint arXiv:2304.08244*, 2023.

[3] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, Y. Huang, C. Xiao, C. Han *et al.*, "Tool learning with foundation models," *arXiv preprint arXiv:2304.08354*, 2023.

[4] S. Gravitas. (2023) Auto-gpt: An autonomous gpt-4 experiment. [Online]. Available: https://github.com/Significant-Gravitas/Auto-GPT

[5] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[6] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," March 2023. [Online]. Available: https://vicuna.lmsys.org

[7] T. Computer, "Openchatkit: An open toolkit and base model for dialogue-style applications," 3 2023. [Online]. Available: https://github.com/togethercomputer/OpenChatKit

[8] O.-A. Team, "Laion: Open-assistant," April 2023. [Online]. Available: https://open-assistant.io/chat

[9] Databricks, "Databricks' dolly, a large language model trained on the databricks machine learning platform," 3 2023. [Online]. Available: https://github.com/databrickslabs/dolly

[10] Bloomberg. (2023) Samsung bans staff's ai use after spotting chatgpt data leak. [Online]. Available: https://www.bloomberg.com/news/articles/2023-05-02/samsung-bans-chatgpt-and-other-generative-ai-use-by-staff-after-leak#xj4y7vzkg

[11] CNN. (2023) Jpmorgan restricts employee use of chatgpt. [Online]. Available: https://www.cnn.com/2023/02/22/tech/jpmorgan-chatgpt-employees/index.html

[12] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar *et al.*, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[13] OpenAI, "GPT-4 technical report," Mar. 2023.

[14] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[15] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon *et al.*, "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.

[16] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*. PMLR, 2022, pp. 2206–2240.

[17] H. Li, Y. Su, D. Cai, Y. Wang, and L. Liu, "A survey on retrieval-augmented text generation," *arXiv preprint arXiv:2202.01110*, 2022.

[18] O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown, and Y. Shoham, "In-context retrieval-augmented language models," *arXiv preprint arXiv:2302.00083*, 2023.

[19] A. Glaese, N. McAleese, M. Trębacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick, P. Thacker *et al.*, "Improving alignment of dialogue agents via targeted human judgements," *arXiv preprint arXiv:2209.14375*, 2022.

[20] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," *arXiv preprint arXiv:2209.07753*, 2022.

[21] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, Jan 2023.

[22] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[23] V. Sanh, A. Webson, C. Raffel, S. H. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, T. L. Scao, A. Raja *et al.*, "Multitask prompted training enables zero-shot task generalization," *arXiv preprint arXiv:2110.08207*, 2021.

[24] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[26] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave, "Few-shot learning with retrieval augmented language models," *arXiv preprint arXiv:2208.03299*, 2022.

[27] D. GmbH. (2023) Haystack documentation. [Online]. Available: https://docs.haystack.deepset.ai/docs/retriever#bm25-recommended

[28] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[29] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," 2018.

[30] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[31] S. Yao, H. Chen, J. Yang, and K. Narasimhan, "Webshop: Towards scalable real-world web interaction with grounded language agents," 2023.

[32] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[33] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[34] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[35] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: A compiler infrastructure for the end of moore's law," *arXiv preprint arXiv:2002.11054*, 2020.

[36] L. B. Allal, R. Li, D. Kocetkov, C. Mou, C. Akiki, C. M. Ferrandis, N. Muennighoff, M. Mishra, A. Gu, M. Dey *et al.*, "Santacoder: don't reach for the stars!" *arXiv preprint arXiv:2301.03988*, 2023.

[37] S. Yang, O. Nachum, Y. Du, J. Wei, P. Abbeel, and D. Schuurmans, "Foundation models for decision making: Problems, methods, and opportunities," *arXiv preprint arXiv:2303.04129*, 2023.

[38] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz *et al.*, "Augmented language models: a survey," *arXiv preprint arXiv:2302.07842*, 2023.

[39] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[40] Y. Liang, C. Wu, T. Song, W. Wu, Y. Xia, Y. Liu, Y. Ou, S. Lu, L. Ji, S. Mao *et al.*, "Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis," *arXiv preprint arXiv:2303.16434*, 2023.

[41] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[42] A. Parisi, Y. Zhao, and N. Fiedel, "Talm: Tool augmented language models," *arXiv preprint arXiv:2205.12255*, 2022.

[43] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[44] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," *arXiv preprint arXiv:2209.11302*, 2022.

[45] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *2023*, 2023.

[46] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[47] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, "Visual chatgpt: Talking, drawing and editing with visual foundation models," *arXiv preprint arXiv:2303.04671*, 2023.

[48] G. Kim, P. Baldi, and S. McAleer, "Language models can solve computer tasks," *arXiv preprint arXiv:2303.17491*, 2023.

[49] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," in *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, vol. 11, no. 3. NIH Public Access, 2017, p. 269.

[50] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," 2023.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] N. Muennighoff, T. Wang, L. Sutawika, A. Roberts, S. Biderman, T. L. Scao, M. S. Bari, S. Shen, Z.-X. Yong, H. Schoelkopf *et al.*, "Crosslingual generalization through multitask finetuning," *arXiv preprint arXiv:2211.01786*, 2022.

[53] BigScience Workshop, "BLOOM (revision 4ab0472)," 2022. [Online]. Available: https://huggingface.co/bigscience/bloom

[54] Chavez. (2023) chavinlo/gpt4-x-alpaca. [Online]. Available: https://huggingface.co/chavinlo/gpt4-x-alpaca

[55] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[56] S. Iyer, X. V. Lin, R. Pasunuru, T. Mihaylov, D. Simig, P. Yu, K. Shuster, T. Wang, Q. Liu, P. S. Koura *et al.*, "Opt-iml: Scaling language model instruction meta learning through the lens of generalization," *arXiv preprint arXiv:2212.12017*, 2022.

[57] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[58] A. Andonian, Q. Anthony, S. Biderman, S. Black, P. Gali, L. Gao, E. Hallahan, J. Levy-Kramer, C. Leahy, L. Nestler, K. Parker, M. Pieler, S. Purohit, T. Songz, W. Phil, and S. Weinbach, "GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch," 8 2021. [Online]. Available: https://www.github.com/eleutherai/gpt-neox

[59] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[60] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang *et al.*, "Gpt-neox-20b: An open-source autoregressive language model," *arXiv preprint arXiv:2204.06745*, 2022.

[61] Together, LAION, and Ontocord.ai. (2023) The oig dataset. [Online]. Available: https://huggingface.co/datasets/laion/OIG

[62] S. Biderman, H. Schoelkopf, Q. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," *arXiv preprint arXiv:2304.01373*, 2023.

[63] Databricks, "dolly-v2-12b," 2023. [Online]. Available: https://huggingface.co/databricks/dolly-v2-12b

[64] Stablility-AI. (2023) Stablelm. [Online]. Available: https://github.com/Stability-AI/StableLM

[65] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[66] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis *et al.*, "Spatial: A language and compiler for application accelerators," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2018, pp. 296–311.

[67] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 389–402, 2017.

[68] R. Prabhakar and S. Jairath, "Sambanova sn10 rdu: Accelerating software 2.0 with dataflow," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–37.

[69] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[70] J. Hauke and T. Kossowski, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones geographicae*, vol. 30, no. 2, pp. 87–93, 2011.

In the appendix section, we provide detailed information on the following aspects of our study. In Appendix A, we present the background and curation details for the 8 tasks included in ToolBench. Appendix B focuses on the performance evaluation of an extensive suite of LLMs on ToolBench. In Appendix C, we delve into the details of model alignment, including the process of generating the training data and training details. We also provided the full spectrum of results for the experiments in Section 6. Finally, in Appendix D, we introduce the API selection complexity score system, and demonstrate its effectiveness and implication in measuring task complexity.

A Benchmark Details

A.1 OpenWeather

This task involves using the REST API to interact with OpenWeather website[7]. We include 9 types of API calls that cater to 9 categories of queries, including but not limited to retrieving current weather data in a city, obtaining air quality data at a specific longitude and latitude, and acquiring weather forecast data for a location specified by a zip code. Making each type of API calls involves correctly filling 2 to 3 required parameters (such as `lon` for longitude and `lat` for latitude) and 0 to 3 optional parameters (such as `lang` for language and `units` for units of measurement), depending on the requirements specified in each query. In total, we develop 100 unique queries for the 9 categories and 2 demonstration examples for each category. To assess the quality of the LLM's generation, we look for the first line beginning with the word "curl", if it exists. We then execute this line using the shell process. If the shell process returns a non-zero value, we declare "not executable" for this generation. On the other hand, if the code can be executed, we compare the returned response with the corresponding result from the ground-truth Curl request. The model's generation will be considered successful if the output matches the expected result precisely.

A.2 The Cat API

This task is a similar REST API task as the OpenWeather, but it involves making all the `GET`, `DELETE`, or `POST` request to The Cat API website[8]. There are 6 types of API calls for 6 types of queries, including deleting a cat image from the user's list of favorites, adding an image to the user's list of favorites, returning the list of favorite images, voting up or down to an image, and searching for cat images with filtering requirements. We develop 100 queries for the test set and 2 demonstration examples for each category. To evaluate the executability and success of the LLM's generation in these scenarios, we follow a similar procedure as that of the Open Weather task. It is worth noting that for queries related to removing an image from the list of favorites, we compare the LLM's generation verbatim with the ground-truth label since duplicated deletion would inevitably lead to failure if executed.

A.3 Home Search

This task is designed to replicate the process of searching for homes at a specific location based on certain criteria. We design the API with 15 functions, including

- `set_location` which sets the desired location;
- `set_but_or_rent` which specifies whether the user is looking to buy or rent a home;
- 12 functions for setting criteria, such as home prices, number of bedrooms, and home square footage;
- `search` which submits the criteria to get search results.

We consider executability and f1 score of the generated action. To ensure executable searches, the agent should make a sequence of function calls that starts with `set_location` and `set_buy_or_rent`, followed by the criterion-setting functions, and then ends with a call to the `search` function. If executable, an f1 score is computed between the criteria set by the generated program and that by the ground-truth program. We develop a test set consisting of 100 queries that

[7] https://openweathermap.org/api

[8] https://thecatapi.com

asked for home options with varying criteria combinations and provide 10 demonstration examples. To test the LLM's ability to utilize unseen API functions, we intentionally exclude 3 criterion-setting functions from all demonstration examples.

A.4 Trip Booking

The Trip Booking task is similar to the Home Search task but with more advanced dependency requirements among function calls. It simulates the process of submitting search requests for transportation tickets, hotel rooms, or both based on specific requirements like locations, dates, and the number of tickets required. We design 20 functions for the three types of booking scenarios. Depending on the scenario, some function calls may be required while others are optional. Missing any required function call or mistake the order of some function calls results in a non-executable search, while missing optional function calls lead to an unsuccessful search. We include 120 queries in the test set and provide 11 demonstration examples.

A.5 Google Sheets

This task is to manipulate the real worksheets from the Google Sheets[9], via the gspread library[10]. We include 100 distinct API function calls from the gspread library, but we only create tests for the most common use cases, including updating cell values, sorting, adding or deleting rows and columns, merging cells, filtering, formatting and creating pivot tables. There are 70 test cases and 10 examples in total. We also encourage the model to utilize Pandas DataFrame[11] and gspread-dataframe[12] for advanced manipulations, by explicitly providing 8 additional API functions and certain examples for them. The manipulation is considered as correct only if both the value and the format of each cell match the expectation.

A.6 Virtual Home

This task is inherited from the setting of the VirtualHome[13] simulator and asks the LLM to generate sequences of actions for completing household activities. We develop API definitions, demonstration examples, and a test set based on the list of available examples[14] curated in (30). The API consists of 40 functions, each of which corresponds to a specific action used in the examples. These functions can take up to two arguments, and we collect the list of valid object names for each argument based on all examples. Some examples of the functions include `Sleep()`, `Push(object)`, and `PourInto(object1, object2)`.

The original example list contains 202 household activities, represented by 5088 examples, with each example being a series of actions to complete a specific activity. However, some activities have exactly the same solution as another activity. After deduplication, we are left with 183 unique activities with non-overlapping solutions between any two activities. We randomly select 100 activities to form the test set, while the remaining 83 tasks with their 512 solutions are used as demonstration examples.

When evaluating the LLM's generation for a given task, we consider both executability and correctness. The generation is considered executable if it can be correctly parsed into a series of valid actions, where each action involves only recognizable objects. Regarding correctness, we measure the similarity between the generated program and the ground-truth solution, using the longest common subsequence (LCS) (29) normalized by the maximum length of the two. For tasks with multiple solutions, we consider the highest LCS score from any solution.

A.7 WebShop

This is a multi-step task inherited from Webshop (31), a simulated online shopping environment. The task requires an agent to navigate through a series of webpages to find and purchase a desired product

[9] https://www.google.com/sheets/about/

[10] https://docs.gspread.org/

[11] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[12] https://gspread-dataframe.readthedocs.io/en/latest/

[13] http://virtual-home.org/

[14] https://github.com/huangwl18/language-planner/blob/main/src/available_examples.json

based on a text instruction that outlines the item description.The agent can perform two primary types of actions: `search[text]`, which involves entering a text query, and `click[button]` which involves selecting a button on the page.

We generate demonstration examples based on this file[15], which contains trajectories collected from humans performing the online shopping tasks. We formulate each trajectory into a series of (instruction, webpage description, action) tuples in plain text format. The Long version of the demonstration set consists of 1533 full trajectories, which often exceed the input sequence length limit of the LLM. To address this issue, we provide a Short version of the demonstration examples, by first removing 80% of the non-targeted items from any webpage description, and selecting only the 200 shortest trajectories from the complete set.

For evaluation, we use the predefined simple mode of the WebShop environment[16] and set up the environment with the provided option of using only 1000 random products. We include 100 instructions from sessions with ID numbers 0 to 99 in the test set. We define success as making a purchase which receives a positive reward from the environment within 25 steps.

A.8 Tabletop

This task is developed based on the simulated tabletop manipulation domain presented by (50) and outlined in their Appendix K. In this simulation environment, a UR5e robot with a Robotiq 2F85 jaw gripper can perform pick and place actions parameterized by 2D top-down positions. We reuse their API definitions and prompts as demonstration examples. We iterate on the 14 instruction templates used in their evaluation benchmark and create 15 types of tasks that involve manipulating up to 4 colored blocks and 4 colored bowls. For each type of task, we generate 7 valid initial setups of blocks and bowls for the test set, ensuring that no collisions occur during the execution of a valid solution. The success of the LLM's generated program is determined by whether all objects are within a small threshold of their target positions after execution.

B Comprehensive Model Evaluation on the ToolBench

In this section, we want to compare the performance of different models on the ToolBench. Specifically, we selected 27 representative LLMs from both closed and open-source community, and evaluate them on the ToolBench in 3-shot scenario.

B.1 Models

As listed in Table 8, we select a set of representative LLMs from both closed-source and open-source community.

The closed models are the (Generative Pre-trained Transformer) GPT series from OpenAI, especially the GPT-3(22) and its successors(13). GPT-3 is a state-of-the-art language model developed by OpenAI, with 175 billion parameters, making it the largest and most powerful language model ever created. It is capable of performing a wide range of natural language processing tasks and has the potential to revolutionize the way we interact with and understand language. Due to the lack of detailed information about its training, we are motivated to study methods to build models achieving similar capabilities, especially using open-source models.

We select the representative and the most advanced open-source models from recent years in our work. They are all decoder-only models, based on transformers(51) architecture. Bloomz(52) is the largest open-source LLM built upon the large-scale multilingual pretrained BLOOM(53). Bloomz is funtuned on xP3(52), a crosslingual task mixture, for crosslingual generalization to unseen tasks and languages. StarCoder(33) is a family of models developed for purely code generation and synthesis with 8K context length. They exhibit superior performance on common code generation benchmarks. LLaMA(32) is a family of pretrained models, that are performant on quite a few NLP benchamrks. Although they are not as large as Bloomz, they are all trained for almost 4 times longer than Bloom. This is an important reason why they are able to outperform several top peer models on many NLP tasks. Alpaca (54; 55) is fine-tuned LLaMA-13b model on 52K instruction-following data as well

[15] https://github.com/princeton-nlp/WebShop/blob/master/baseline_models/data/il_trajs_finalized_images.zip
[16] https://github.com/princeton-nlp/WebShop#text-environment-simple-mode

Table 8: The achitecture and training data of all the models in our evaluation. The models are grouped by their architecture and training data.

Model	Architecture			Data		
	Family	Size	Max SS	# Tokens	Pretraining	Finetuning
Closed-source						
text-davinci-003	gpt3	175b	4096	-	-	-
gpt-3.5-turbo	gpt3	-	4096	-	-	-
text-curie-001	gpt3	6.7b	2048	-	-	-
gpt4	gpt4	-	8192	-	-	-
Open-source						
bloomz	bloom	176b	2048	366B	bloom corpus	xP3
llama-65b	llama	65b	2048	1.4T	CCNet, C4,	-
llama-30b	llama	30b	2048	1.4T	GitHub, Wikipedia,	-
llama-13b	llama	13b	2048	1.4T	Books, ArXiv,	-
llama-13b-alpaca	llama	13b	2048	1.4T	Stack Exchange	GPT-4 responses, Alpaca
starcoderbase	bigcode	15.5b	8192	1T	The Stack	-
starcoder	bigcode	15.5b	8192	1T		The Stack (Python)
opt-30b	opt	30b	2048	300B	The Pile, BookCorpus,	-
opt-1.3b	opt	1.3b	2048	300B	CC-Stories, Reddit,	-
opt-iml-30b	opt	30b	2048	300B	CCNewsV2	OPT-IML Bench
opt-iml-1.3b	opt	1.3b	2048	300B		
gpt-neox-20b	neox	20b	2048	450B	The Pile	-
GPT-NeoXT-Chat-Base-20B	neox	20b	2048	460B		OpenChatKit IT
codegen-16B-nl	neox	16b	2048	700B		-
codegen-16B-multi	neox	16b	2048	1T		BigQuery
codegen-16B-mono	neox	16b	2048	1T		BigQuery, BigPython
pythia-12b	neox	12b	2048	300B		-
dolly-v2-12b	neox	12b	2048	300B		Dolly IT
pythia-6.9b/2.8b1.4b	neox	multi	2048	300B		-
stablelm-base-alpha-7b	neox	7b	4096	800B	The Pile (1.5T)	-
stablelm-base-alpha-3b	neox	3b	4096	800B		-
stablelm-tuned-alpha-7b	neox	7b	4096	800B		Alpaca, GPT4All,
stablelm-tuned-alpha-3b	neox	3b	4096	800B		Anthropic, Dolly, ShareGPT

responses from GPT-4. OPT-IML(56) is the finetuned version of the original OPT(57), which is the first family of large-scale (176 billion parameters) open-source models that are trained on publicly available datasets. OPT-IML significatly improves the instruction following capability of OPT by training on a large benchmark of 2000 NLP tasks for Instruction MetaLearning (IML). We only select the publicly accessibile checkpoints from the OPT families in our work.

Another important family of models are all developed from the NeoX toolkit(58) and pretrained using the PILE dataset(59). GPT-NeoX-20B(60) is only pretrained on the PILE, while GPT-NeoXT-Chat-Base-20B(7) is further finetuned on the OIG-43M(61), a dataset targetting better instruction following capability. CodeGen family(34) is designed for superior capability on code generation, as they are heavily finetuned on large code datasets. Pythia family(62) is a suite of models designed for analyzing LLMs across training and scaling. They are all pretrained on the Pile in the same way, but have different model sizes and intermediate checkpoints released during training. We use those variants in our ablation study. Dolly(63) is finetuned beyond Pythia-12b on a new, high-quality human generated instruction following dataset, crowdsourced among Databricks employees. The StableLM family(64) is pre-trained on an experimental version of the PILE datasets which has 1.5 trillion tokens in total. The models have a sequence length of 4096 to push beyond the context window limitations of the existing open-source language models. The instruction tuned counterpart of each model is also released. By the time we publish this work, only 7b and 3b models are released, while the team behind them is training larger models.

There are other notable models, such as FlanT5(28), the T0 family(23), and the T5 family(65), that have shown promising performance. We do not include all of them in our baseline comparison, as some of their features are not designed for the task at hand. For example, their tokenizers do not distinguish between spaces, tabs and new lines, making it hard for them to generate executable code based on API function calls.

Table 9: The performance on ToolBench of different models in 3-shot scenario. The models are group by their architecture and training data.

Task	Open Weather	The Cat API	Home Search	Trip Booking	Google Sheets	VirtualHome	WebShop Long	WebShop Short	Tabletop
max tokens to generate	128	128	128	300	256	128	128		256
num API function	all	all	all	all	10	10	all		0
Closed-source									
gpt4	93.0	96.0	97.0	96.7	62.9	23.0 / 23.5	0.0		81.0
text-davinci-003	99.0	98.0	97.0	89.2	62.9	31.0 / 25.1	0.0		66.7
gpt-3.5-turbo	90.0	92.0	80.0	85.8	51.4	20.0 / 18.9	0.0	1.8	33.3
text-curie-001	8.0	58.0	6.0	6.7	1.4	12.0 / 4.1	0.0	0.0	1.0
Open-source									
llama-65b	90.0	80.0	84.0	65.8	32.9	32.0 / 20.3	0.0	41.2	30.5
llama-30b	78.0	84.0	66.0	45.0	37.1	27.0 / 21.7	0.0	30.6	34.3
llama-13b	70.0	74.0	45.0	35.8	5.7	28.0 / 18.9	0.0	27.6	17.1
llama-13b-alpaca	62.0	43.0	44.0	40.8	11.4	1.0 / 1.6	0.0	2.7	9.5
starcoder	91.0	84.0	82.0	51.7	48.0	23.0 / 19.4	2.6	0.0	21.9
starcoderbase	90.0	86.0	79.0	63.3	42.9	24.0 / 16.3	5.8	23.1	17.1
codegen-16B-nl	51.0	75.0	37.0	21.7	7.1	43.0 / 18.0	0.0	0.0	16.2
codegen-16B-multi	56.0	75.0	47.0	7.5	21.4	31.0 / 14.1	0.0	0.5	8.6
codegen-16B-mono	63.7	72.0	52.0	28.3	31.5	28.0 / 15.7	1.5	6.6	15.2
bloomz	58.0	85.0	36.0	22.5	14.3	9.0 / 4.9	0.0	1.0	1.0
opt-iml-30b	44.0	48.0	5.0	3.3	2.9	13.0 / 8.3	0.0	0.0	1.0
opt-30b	46.0	35.0	2.0	3.3	8.6	24.0 / 11.7	0.0	0.0	1.0
opt-iml-1.3b	20.0	28.0	0.0	0.0	4.3	13.0 / 3.1	0.0	0.0	1.0
opt-1.3b	18.0	30.0	0.0	0.0	1.4	31.0 / 9.7	0.0	0.0	1.0
neox-20b	55.0	69.0	27.0	10.8	18.6	28.0 / 15.3	0.0	8.8	6.7
GPT-NeoXT-Chat-Base-20B	43.0	73.0	28.0	10.8	4.3	26.0 / 13.1	0.0	0.7	7.6
pythia-12b	53.0	65.0	12.0	0.8	11.4	17.0 / 12.1	0.0	0.0	1.9
dolly-v2-12b	0.0	1.0	10.0	5.0	7.1	11.0 / 8.9	0.0	0.0	7.6
pythia-12b	53.0	65.0	12.0	0.8	11.4	17.0 / 12.1	0.0	0.0	1.9
pythia-6.9b	41.0	72.0	8.0	7.5	4.3	29.0 / 14.0	0.0	0.0	8.6
pythia-2.8b	49.0	54.0	7.0	3.3	12.9	24.0 / 14.8	0.0	0.0	7.6
pythia-1.4b	37.0	48.0	4.0	5.0	10.0	22.0 / 10.7	0.0	5.2	7.6
stablelm-base-alpha-7b	22.0	47.0	0.0	0.0	4.3	28.0 / 10.3	0.0	0.0	2.9
stablelm-tuned-alpha-7b	23.0	38.0	0.0	0.0	1.4	26.0 / 7.3	0.0	0.0	3.8
stablelm-base-alpha-3b	6.0	28.0	0.0	0.0	1.4	29.0 / 5.3	0.0	0.0	1.0
stablelm-tuned-alpha-3b	14.0	31.0	0.0	0.8	0.0	8.0 / 5.6	0.0	0.0	1.0

B.2 Evaluation

To collect the baseline results, we exploit the naive approach described in section 2 as the action generator. We give each LLM sufficient max tokens to generate on each task and retrieve as many API functions as possible in the prompt. The detailed information is listed in Table 9. We evaluate all the models on a mixture of GPUs and RDUs(66; 67; 68). In particular, the 176b-parameter bloomz is evaluated on RDU, while all the other models are evaluated on NVIDIA A100 GPUs with 80GB RAM.

For these models, We only conduct the few-shot evaluation described Section 6 because 1) zero-shot results are not representative, as most of them are zero, 2) it is not practical to tune all the models on our training data, and 3) few-shot results can be used as a great proxy of the model performance in all the other settings. For the conversation-oriented models, including gpt-3.5-turbo, chavinlo/gpt4-x-alpaca, GPT-NeoXT-Chat-Base-20B and dolly-v2-12b, we additionally add `<human>:` and `<bot>:` key words in the prompt to better align with their training data format for better performance.

After we get the completion from the LLMs given a prompt, only minimal post-processing steps are applied to the completion: 1) Properly truncate the completion, given the list of task-specific stop sequences and 2) Replace the `{API_KEY}` keywords in the completion with the real API key, so as to execute the code properly. Finally, as shown in Figure 1, to validate the action generated for the single-step tasks, we execute the generated API calls and compare its output against the ground truth; while for the multi-step tasks, the actions are used to interact with the environment directly and only

the final status is evaluated. For each task, we report the metrics described in Section 5 for each task. Note that we only evaluate the top 1 generated action with sampling disabled. This is because, in practice, action can only be executed once and there is no chance to reset things and try another action.

B.3 ToolBench performance of different models

The performance of different models are summarized in Table 9. Below we show several observations.

Capability Gap Currently, the GPT family of models stands out as the leading players in the field, and there is a significant gap between GPT-4, GPT-3.5 and all the other open-source models. While open-source models may demonstrate competitiveness on some simpler tasks, they lag far behind on more challenging tasks such as Google Sheets and Tabletop.

Instruction tuning on conventional NLP tasks doesn't help Comparing the models between chavinlo/gpt4-x-alpaca and LLaMA-13b, OPT-IML and OPT, StableLM-tuned and StableLM-base, NeoX-Chat-Base-20b and NeoX, and dolly and pythia, the former model in each pair is intentionally optimized to enhance instruction following capability compared to the latter model. However, no significant accuracy improvement is observed on the ToolBench. Further, the LLaMA family, despite not undergoing any specific instruction tuning during training, still achieves relatively good quality compared to other public models.

Model size is important By comparing the performance of models from GPT faimily, LLaMA family, OPT family, Pythia family and StableLM family, we can clearly see the trend that the larger models tend to perform better on the ToolBench, given the same quantity and quality of their training data.

Code generation is important StarCoder and CodeGen faimily stand out among other models with similar sizes on ToolBench, while StarCoderBase is even on par with the llama-65b model which is more than 4 times larger in size. CodeGen-16B-mono is overall better than its base model CodeGen-16B-nl, which is not specifically tuned for code generation. It is also surprisingly better than CodeGen-16B-multi on almost all the tasks, indicating that it is highly beneficial for action generation if the model is heavily tuned on Python-style code generation.

C Experiment Details

In this section, we extended Section 6 with more details about model training and results.

C.1 Training data

For the OpenWeather, The Cat API, Trip Booking, and Home Search tasks, we generate the training data by converting or expanding the demonstration examples of each task into templates and populating them with various sets of variable values. For the remaining four tasks, we format the training samples directly from the demonstration example set described in section 5. We exclude any test samples from the training data and minimize the overlap of the API function call combinations between any training and test samples. For example, we make sure that the API function combinations used in each test case for the Home Search task are never present in the training data. However, for the OpenWeather task, it was unavoidable to have some overlap because each test case only involved a single function call and the training examples covered all the API functions. The numbers of templates and training samples for each task are summarized in table 10. Example templates and variable values are shown in table 11. The training sets for all tasks, except for the Google Sheets and WebShop task, reduce the complexity score of their respective test sets when compared to the example sets. As expected, the model's accuracy shows improvement after fine-tuning.

C.2 All-shot loss

To construct the training samples, we concatenate API documents and multiple pairs of goal and API calls as one input sequence to the LLMs. We use an all-shot loss formulation illustrated in Figure 6

Table 10: The statistics of model alignment data

Task	Open Weather	The Cat API	Home Search	Trip Booking	Google Sheets	VirtualHome	WebShop	Tabletop
Templates	90	40	100	30	1	1	2	1
Repeat	20	45	18	60	118	512	900	74
Training samples	1800	1800	1800	1800	118	512	1800	74
Complexity score	1.1	1.0	6.4	10.1	12.1	12.3	0.0	4.6

Table 11: Training template examples of different tools

	Goal	Action	Variable values
Open Weather	What is the present weather situation in {city}? Please respond in {lang} and use {units} units.	curl -X GET 'https://api.openweathermap.org/data/2.5/weather?q={city_formatted}&appid={API_KEY}&lang={lang_abbr}&units={units}'	{city: "Palo Alto", city_formatted: "palo+alto", lang: "English", lang_abbr: "en", units: "imperial"}
The Cat API	Add the cat photo with id={image_id} to my list of favorites.	curl -X POST 'https://api.thecatapi.com/v1/favourites' –data '{"image_id":"{image_id}"}'	{image_id: "MTUyNTA1OA"}
Home Search	Looking for homes for sale in {location} with {num_beds} bedrooms and {num_baths} bathrooms, between ${min_price} and ${max_price}.	API.set_location({location}) API.set_buy_or_rent("buy") API.set_num_beds({num_beds}) API.set_num_baths({num_baths}) API.set_min_price({min_price}) API.set_max_price({max_price}) API.search()	{location: "Palo Alto", num_beds: 4, num_baths: 5, min_price: 7000000, max_price: 8000000}
Trip Booking	Search for {means_of_transportation} tickets for {num_adults} adults from {location_from} to {location_to}, on {departure_date}.	API.select_booking_type("trip tickets") API.select_transportation({means_of_transportation}) API.set_num_adults({num_adults}) API.set_origin(Loc({location_from})) API.set_destination(Loc({location_to})) date = Date({departure_date}) API.set_departure_date(date) API.search()	{means_of_transportation: "flight", max_price_ticket: 150, num_adults: 2, location_from: "San Francisco", location_to: "Los Angeles", departure_date: "2023-08-15"}
Google Sheet	{task}	{action}	{task: " | Product | Cost | Price | | beef | 1 | 3 | | pork | 5 | 4 | | chicken | 10 | 11 | | lamb | 3 | 15 | | duck | 12 | 2 | | fish | 2 | 100 | Task: Sum B1:B4 and write the result below B4 Action:", action: " worksheet.update('B5', '=SUM(B1:B4)', raw=False)"}
VirtualHome	{task}	{action}	{task: " Task: Read book Action:", action: " Agent.Find(novel) Agent.Grab(novel) Agent.Find(chair) Agent.SitOn(chair) Agent.Read(novel)"}
WebShop	{task}	{action}	{task: "Instruction: i'm looking to buy a high resolution marine animal themed backdrop. the size should be 12x10ft, and price lower than 100.00 dollars [button] Search [button_] Action:", action: " search[12x10ft high resolution marine animal backdrop]"}
Tabletop	{task}	{action}	{task: "objects = ['yellow block', 'green block', 'yellow bowl', 'blue block', 'blue bowl', 'green bowl'] # move the green block to the top right corner.", action: " corner_pos = parse_position('top right corner') put_first_on_second('green block', corner_pos)"}

Figure 6: We use all-shot loss for model alignment. We concatenate several examples into a single training sample and backpropagate through the loss on the blue actions in every example. There is no separator token between examples.

Table 12: The detailed performance on the ToolBench of models with different techniques applied. Mean(standard deviation) values are provided for each task. There exists some inevitable randomness, but it won't cange the results by too much.

Task	Open Weather	The Cat API	Home Search	Trip Booking	Google Sheets	VirtualHome	WebShop Long	WebShop Short	Tabletop
Zero-shot Baseline									
gpt4	81.3(1.7)	97.4(0.3)	76.6(1.1)	91.5(0.5)	5.7(0.0)	40.8(0.6) / 8.0(0.2)	0.0(0.0)	-	-
llama-30b	39.0(0.0)	49.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	78.0(0.0) / 0.3(0.0)	0.0(0.0)	-	-
starcoder	32.0(0.0)	71.0(0.0)	7.0(0.0)	13.3(0.0)	5.9(1.1)	22.0(0.0) / 3.7(0.0)	0.0(0.0)	-	-
codegen-16B-mono	7.0(0.0)	78.0(0.0)	0.0(0.0)	0.0(0.0)	1.4(0.0)	4.0(0.0) / 1.0(0.0)	0.0(0.0)	-	-
Sys. Prompt									
gpt4	78.4(0.3)	94.2(0.8)	72.7(2.0)	89.6(0.9)	28.6(0.0)	42.8(0.6) / 8.6(0.1)	0.0(0.0)	-	-
llama-30b	50.0(0.0)	88.0(0.0)	0.0(0.0)	0.0(0.0)	11.4(0.0)	24.0(0.0) / 2.5(0.0)	0.0(0.0)	-	-
starcoder	71.0(0.0)	91.0(0.0)	2.0(0.0)	7.5(0.0)	15.9(0.2)	26.0(0.0) / 4.9(0.0)	0.0(0.0)	-	-
codegen-16B-mono	32.0(0.0)	69.0(0.0)	0.0(0.0)	0.0(0.0)	7.1(0.0)	5.0(0.0) / 1.6(0.0)	0.0(0.0)	-	-
3-shot									
gpt4	93.0(0.0)	96.0(0.0)	97.0(0.0)	96.7(0.0)	62.9(0.0)	23.0(0.0) / 23.5(0.0)	0.0(0.0)	0.0(0.0)	81.0(0.0)
llama-30b	78.0(0.0)	84.0(0.0)	66.0(0.0)	45.0(0.0)	37.1(0.0)	27.0(0.0) / 21.7(0.0)	0.0(0.0)	30.6(0.0)	34.3(0.0)
starcoder	91.0(0.0)	84.0(0.0)	82.0(0.0)	51.7(0.0)	48.0(1.1)	23.0(0.0) / 19.4(0.0)	2.6(0.0)	0.0(0.0)	21.9(0.0)
codegen-16B-mono	63.7(0.5)	72.0(0.0)	52.0(0.0)	28.3(0.0)	31.5(0.5)	28.0(0.0) / 15.7(0.0)	1.5(0.0)	6.6(0.0)	15.2(0.0)
Alignment									
llama-30b	100.0(0.0)	94.0(0.0)	85.0(0.0)	87.5(0.0)	4.3(0.0)	14.0(0.0) / 10.6(0.0)	20.8(0.0)	-	-
starcoder	95.0(0.0)	98.0(0.0)	78.0(0.0)	85.0(0.0)	10.0(0.0)	28.0(0.0) / 13.4(0.0)	0.0(0.0)	-	-
codegen-16B-mono	99.0(0.0)	95.8(0.6)	78.0(0.0)	73.3(0.0)	10.0(0.0)	10.0(0.0) / 11.5(0.0)	30.3(0.0)	-	-
Sys. Prompt + 3-shot									-
gpt4	99.0(0.0)	98.0(0.0)	98.0(0.0)	99.2(0.0)	68.6(0.0)	29.0(0.0) / 21.7(0.0)	0.0(0.0)	0.0(0.0)	83.8(0.0)
llama-30b	66.0(0.0)	82.0(0.0)	63.0(0.0)	45.8(0.0)	27.1(0.0)	34.0(0.0) / 20.5(0.0)	0.0(0.0)	0.0(0.0)	34.6(0.2)
starcoder	92.0(0.0)	91.0(0.0)	73.0(0.0)	54.2(0.0)	50.0(0.2)	28.0(0.0) / 15.0(0.0)	0.0(0.0)	0.0(0.0)	23.4(0.3)
codegen-16B-mono	64.2(0.3)	70.0(0.0)	45.0(0.0)	22.5(0.0)	28.6(0.9)	27.0(0.0) / 15.7(0.0)	0.0(0.0)	0.0(0.0)	14.6(0.2)
Sys. Prompt + Alignment									
llama-30b	100.0(0.0)	94.0(0.0)	79.0(0.0)	80.8(0.0)	5.7(0.0)	10.0(0.0) / 10.3(0.0)	0.6(0.0)	-	-
starcoder	98.7(0.2)	97.0(0.0)	79.0(0.0)	84.2(0.0)	10.0(0.0)	18.0(0.0) / 10.3(0.0)	0.0(0.0)	-	-
codegen-16B-mono	99.0(0.0)	96.0(0.0)	77.0(0.0)	75.8(0.0)	8.6(0.0)	7.0(0.0) / 10.0(0.0)	25.7(0.0)	-	-
3-shot + Alignment									
llama-30b	100.0(0.0)	94.0(0.0)	88.0(0.0)	89.2(0.0)	4.3(0.0)	20.0(0.0) / 26.3(0.0)	19.5(0.0)	15.1(0.0)	6.9(0.2)
starcoder	100.0(0.0)	96.0(0.0)	91.0(0.0)	84.2(0.0)	15.7(0.0)	48.0(0.0) / 21.3(0.0)	0.0(0.0)	0.0(0.0)	13.9(0.3)
codegen-16B-mono	99.0(0.0)	97.9(0.2)	80.0(0.0)	77.5(0.0)	16.4(0.9)	38.0(0.0) / 18.6(0.0)	6.5(0.0)	17.5(0.0)	16.2(0.0)
Prompt + 3-shot + Alignment									
llama-30b	100.0(0.0)	94.0(0.0)	87.0(0.0)	85.8(0.0)	2.9(0.0)	16.0(0.0) / 24.3(0.0)	0.0(0.0)	0.0(0.0)	7.5(0.1)
starcoder	99.0(0.0)	97.0(0.0)	83.0(0.0)	80.8(0.0)	21.2(0.3)	31.0(0.0) / 18.4(0.0)	0.0(0.0)	0.0(0.0)	13.9(0.3)
codegen-16B-mono	97.7(0.2)	99.0(0.0)	82.0(0.0)	77.5(0.0)	19.8(0.3)	29.0(0.0) / 17.2(0.0)	0.0(0.0)	3.5(0.0)	16.2(0.0)

which learns to generate the API calls for every goal in a sequence. We use this loss formulation because it empirically delivers better success rate, especially when using in-context demonstrations, than the conventional loss which only backpropagates the loss associated with the API calls for the last goal.

C.3 Training details

We finetune each model on the same dataset created with the method described in Section C.1 for 8 epochs. We use a max sequence length of 2048 without packing and mix the data from all the tasks into a single dataset with random shuffling. In each sample, all the goal-action pairs are from the same task. We report the validation accuracy on the best checkpoint. We use a batch size of 16 and a constant learning rate of $1e-5$ for each model and train on an internal cluster of 4 A100 GPU's, each with 80GB RAM.

C.4 Extended results for Section 6

We list out the detailed results of Section 6 in Table 12, where we report the model performance on all the possible combinations of the three proposed techniques. The main observations are all covered in Section 6. We run each job 3 times, and report the mean and standard deviation of the main metrics. Their are some inevitable randomness happens in API or example retrieval, public API services and the environment provided in Webshop and Tabletop. Even though randoness exists, we observe that they barely change the final results. Thus, we only report the mean value everywhere else in the paper.

D API Selection Complexity Score

D.1 Complexity score

This section introduces a complexity score system designed to measure the intrinsic complexity and difficulty of the tasks from *ToolBench*. The complexity score system aims to provide a quantitative measure of the intrinsic complexity of the tests given the examples by calculating the probability of the tests being derived or converted from the examples; and the derivation or conversion is performed in a random system with all possible outcomes equally likely. This score serves to assess the inherent level of difficulty involved in transitioning from one scenario to another, thereby assisting researchers and developers in benchmark evaluation and analysis.

D.1.1 The likelihood of a test being derived from an example

In the complexity score system proposed herein, the calculation of the complexity score involves assessing the probability or likelihood of the tests being derived from an example in the particular task. Given a demonstration example e and a set of API functions $\mathcal{D}$, the derivation of a particular test sample t involves two major steps: 1) remove all the unused API calls while keeping all the necessary ones and 2) insert the new API calls that e does not cover. Given a random system, where all possible outcomes are equally likely, we suppose the deletion possibility of each API call from e is 50%, while the insertion possibilities of the correct API call is $1/|\mathcal{D}|$, where $|\mathcal{D}|$ is the total number of API functions of the given task. If t or e contains multiple calls to the same API function, we consider them as different API calls, because they are usually not interchangeable. Based on these assumptions, the likelihood of generating a test sample t is calculated using Equation (1).

$$p(t \mid e, \mathcal{D}) = \left(\frac{1}{2}\right)^{|e|} \left(\frac{1}{|\mathcal{D}|}\right)^{|t \setminus e|} \tag{2}$$

where $|e|$ represents the number of API calls in the example e, and $|t \setminus e|$ is the number of uncovered API calls in the test sample. Suppose we have a task that has 10 API functions in total $\{a_i\}_1^{10}$, and the demonstration example covers $\{a_1, a_2, a_3, a_4\}$, but the test sample requires $\{a_1, a_2, a_6, a_4, a_5\}$. In the first step, the probability of successfully dropping a_3 while keeping the rest ones in e is $\left(\frac{1}{2}\right)^4$. Then, the probability of correctly adding in the uncovered ones, a_5 and a_6, is $\left(\frac{1}{10}\right)^2$. Note that we do not take the order of API calls into consideration for the purpose of being simple without losing generosity.

D.1.2 The distance between a test and example pair

We first define the distance d between one particular test and example pair by take the logarithm of the reciprocal of Equation (1) as:

$$d(t, e) = \log\left[\frac{1}{p(t \mid e, \mathcal{D})}\right] \tag{3}$$

The use of the reciprocal in the expression aligns the complexity score with the definition of complexity, where a higher score indicates a greater level of complexity. Additionally, applying the logarithm to the reciprocal value aids in addressing the magnitude gap. The logarithm function compresses the range of values, reducing the impact of extreme values and creating a more manageable scale. This normalization ensures that the complexity score is not disproportionately influenced by outliers or

extreme values, providing a more balanced representation of complexity across the range of input values. By combining the reciprocal and logarithm, the expression effectively balances the score by aligning it with the definition of complexity and mitigating the impact of magnitude differences in the input values.

D.1.3 Complexity score of a task

Based on the complexity score of generating a test from an example, we can construct the complexity score S of a given task. The score $S = f(\mathcal{T}, \mathcal{X}, \mathcal{D})$ is a function of the test samples $\mathcal{T}$, the demonstration examples $\mathcal{X}$ and the API functions $\mathcal{D}$ of each task.

$$\begin{aligned} S(\mathcal{T}, \mathcal{X}, \mathcal{D}) &= \mathbb{E}_{t \in \mathcal{T}} \min_{e \in \mathcal{X}} d(t, e) \\ &= \mathbb{E}_{t \in \mathcal{T}} \min_{e \in \mathcal{X}} \log \left[\frac{1}{p(t \mid e, \mathcal{D})} \right] \\ &= -\mathbb{E}_{t \in \mathcal{T}} \max_{e \in \mathcal{X}} \log \left[\left(\frac{1}{2} \right)^{|e|} \left(\frac{1}{|\mathcal{D}|} \right)^{|t \setminus e|} \right] \end{aligned} \tag{4}$$

This score ranges from zero to infinity. The larger the score is, the more challenging a task is in terms of API selection. We calculate this score for both the original ToolBench (Table 4) and the training data we created for alignment Table 10. They share the same $\mathcal{D}$ and $\mathcal{T}$, but have a different $\mathcal{X}$, so that their API selection complexities are different for each task.

D.2 Complexity score on the ToolBench

In this section we demonstrate how the complexity score behaves on the ToolBench.

D.2.1 Computation details

For the Trip Booking, Home Search, Virtual Home, and Google Sheets tasks, the set of API functions $\mathcal{D}$ is the same as described in appendix A. For the single-step, single-API-call tasks, Open Weather and The Cat API, each valid URL with parameters is treated as a unique API option in set $\mathcal{D}$. In total, Open Weather has 37 API options, while The Cat API has 52 API options. In the case of the Tabletop task, since there are no predefined correct answers for the test cases, we divide the three set of "Tabletop Manipulation" examples[17] into 65 single-step samples. Note that for the WebShop task, since there are only two API functions always covered by the example set, the complexity score is 0 by definition.

D.2.2 Reversed-F1 Score

For comparison purpose, we also consider the simple Reversed-F1 (r-F1) distance d_{r-F1}, derived from the conventional F1 score(69), between one particular test and example pair as

$$d_{r-F1}(t, e) = (1 - F1(t, e)) * 100 \tag{5}$$

We multiply 100 to the score to align with the range of the complexity score defined above. Follow the same definition proposed in appendix D.1.3, we can construct the r-F1 score S_{r-F1} of a given task as:

$$\begin{aligned} S_{r-F1}(\mathcal{T}, \mathcal{X}) &= \mathbb{E}_{t \in \mathcal{T}} \min_{e \in \mathcal{X}} d_{r-F1}(t, e) \\ &= \mathbb{E}_{t \in \mathcal{T}} \min_{e \in \mathcal{X}} \left[(1 - F1(t, e)) * 100 \right] \end{aligned} \tag{6}$$

D.2.3 Measurements

[17] https://code-as-policies.github.io/

Figure 7: Spearman's correlation coefficient(SCC) is computed separately for two comparisons: (1) complexity score and error rate, and (2) reversed F1 score and error rate on five tasks: (1) Open Weather, (2) The Cat API, (3) Home Search, (4) Trip Booking, and (5) Virtual Home.

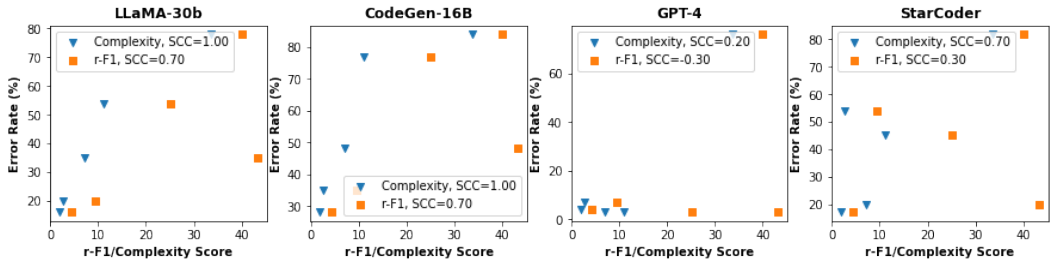


Table 13: Spearman's Correlation Coefficients

	GPT-4	LLaMA	CodeGen	StarCoder
Complexity	0.2	1.0	1.0	0.7
r-F1	-0.3	0.7	0.7	0.3

In this section, Spearman's Correlation Coefficient (SCC) (70) is employed to assess the effectiveness of the proposed complexity score. The evaluation involves the analysis of five different tasks using three models: GPT-4, LLaMA-30b, CodeGen-16b, and StarCoder. We only include the five tasks without advanced reasoning from table 4, as the advanced reasoning breaks the correlation between the API selection difficulty and the final model performance. The complexity score and the r-F1 score are calculated for each task. SCC is then computed separately for two comparisons: (1) complexity score and error rate, and (2) reversed F1 score and error rate, for all five tasks. The results are illustrated in fig. 7 and table 13.

The findings of the study reveal near-perfect Spearman's correlation coefficient (SCC) between the complexity score and the error rate for the LLaMA-30b, CodeGen-16b and StarCoder models. This strong correlation indicates that the proposed complexity score system accurately captures the intrinsic difficulty of these tasks.

For more powerful models like GPT4, which exhibit near-perfect accuracy (above 93%) for low-complexity tasks (complexity < 12) such as Open Weather, The Cat API, Home Search, and Trip Booking, the SCC becomes relatively sensitive to any randomness or turbulence during the experiments. Consequently, the complexity score system shows a non-perfect SCC of 0.2 in this case.

Despite the sensitivity of the SCC in the GPT4 experiments, the complexity score remains a superior indicator of task difficulty compared to the r-F1 score. It effectively captures the inherent difficulty of each task and provides valuable insights into task complexity. Overall, complexity score is more effective at capturing the inherent difficulty of each task, thus providing valuable insights into task complexity.

The obtained results provide empirical evidence supporting the validity and reliability of the proposed complexity score system. The high SCC values signify a consistent relationship between the complexity score and the error rate across different models and tasks. This correlation strengthens the argument that the complexity score accurately captures the complexity and difficulty of the benchmarks, enabling researchers and developers to assess and compare the inherent challenges associated with different tasks.