T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step

Anonymous ACL submission

Abstract

Large language models (LLMs) have achieved remarkable performance on various NLP tasks and are augmented by tools for broader applications. Yet, how to evaluate and analyze the tool utilization capability of LLMs is still under-explored. In contrast to previous works that evaluate models holistically, we comprehensively decompose the tool utilization into multiple sub-processes, including instruction following, planning, reasoning, retrieval, understanding, and review. Based on that, we further introduce T-Eval to evaluate the tool-utilization capability step by step. T-Eval disentangles the tool utilization evaluation into several sub-domains along model capabilities, facilitating the inner understanding of both holistic and isolated competency of LLMs. We conduct extensive experiments on T-Eval and in-depth analysis of various LLMs. T-Eval not only exhibits consistency with the outcome-oriented evaluation but also provides a more fine-grained analysis of the capabilities of LLMs, providing a new perspective in LLM evaluation on tool-utilization ability. The benchmark will be available.

1 Introduction

Large language models (LLMs) have fueled dramatic progress and emerged as a promising path to more advanced intelligence (Zhao et al., 2023; Kaddour et al., 2023). To further extend the capability of LLMs, tool utilization, which empowers LLMs to leverage external tools to solve more complicated problems, has spurred vast research interests in both research and industry (Parisi et al., 2022; Schick et al., 2023; Mialon et al., 2023).

Despite the attractive ability achieved by aiming LLMs with tools, how to evaluate LLMs in tool learning has not been fully explored. Existing works evaluate the tool utilization ability based on the final output (Qin et al., 2023b) or only consider the single-step tool calling (Li et al., 2023b). However, real-world problems usually involve complex planning and executing multiple tools. Simply judging the quality through the final output omits the assessment of the intermediate steps, making it hard to identify the main bottlenecks of the tool-use capability in LLMs. Besides, current benchmarks mainly rely on real-time tool interactions (Qin et al., 2023b; Li et al., 2023b), which overlooks the external factors (instability of API service or temporal information shift) on the overall judgment, leading to evaluation variance and unfair comparison.

To overcome the above problems, we introduce T-Eval, a step-by-step **T**ool **Eval**uation benchmark for LLMs. Unlike prior works that appraise the model from a holistic perspective, we explicitly decompose the evaluation into several sub-tasks along the basic capabilities of the language model. Specifically, given the golden tool-utilization annotations verified by human experts, we dedicatedly designed the evaluation protocols and corresponding instruction prompts based on the intermediate steps along the annotation path. Such a paradigm enables us to separately benchmark each competence of the LLMs, including planning, reasoning, retrieval, understanding, instruction following, and review. Additionally, thanks to the decomposed evaluation protocols, our benchmark significantly alleviates the exogenous influences (such as online tools) during the evaluation process, yielding a more stable and fair model assessment.

By conducting extensive experiments on T-Eval, we carry out in-depth analysis and insights on the results, pinpointing the main bottlenecks of current LLMs in tool learning. Furthermore, we also prove that our benchmark reveals consistent evaluation of individual and comprehensive model abilities, where higher individual ability scores lead to better performance on complex downstream tasks, providing a new perspective in LLM evaluation on tool utilization. Our major contributions are as follows:

- We introduce T-Eval, a step-by-step tool uti-

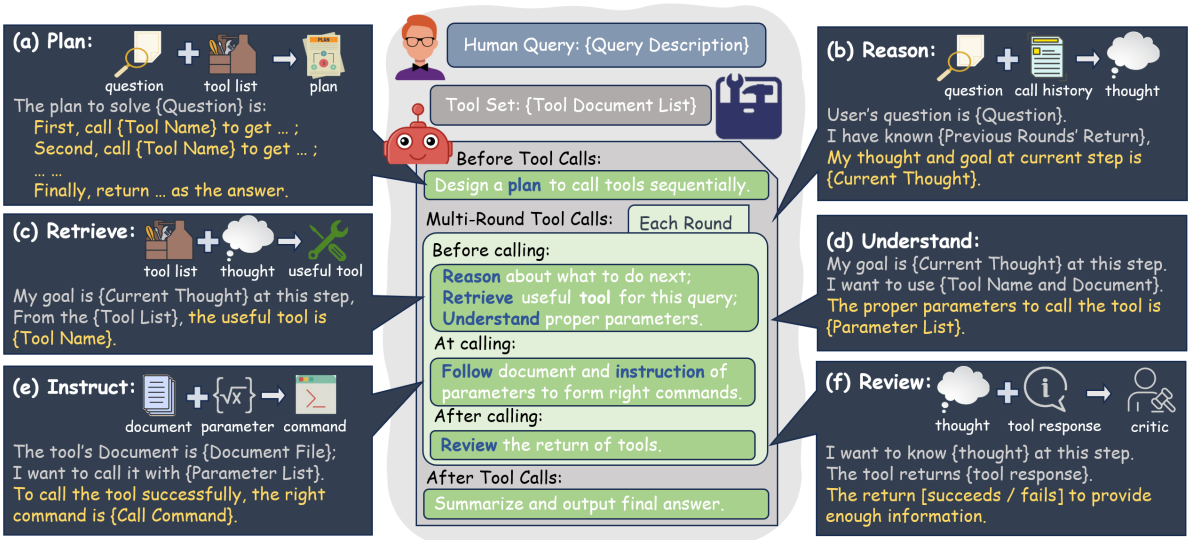


Figure 1: **Overview of `T-Eval`**: `T-Eval` decomposes the tool utilization capability into six necessary abilities: PLAN, REASON, RETRIEVE, UNDERSTAND, INSTRUCT and REVIEW. To respond to a query with a given tool list, LLM agents generate a **plan** first before calling tools. The solution path is multiple rounds of tool calling where agents **reason** their thoughts, **retrieve** and **understand** the necessary tools and parameters, execute the **instructions**, and finally **review** the tool response.

lization evaluation benchmark, which decomposes the evaluation into several sub-tasks, gauging the fine-grained abilities of LLMs as tool agents.

- `T-Eval` uses a multi-agent data generation pipeline verified by human experts. This approach significantly reduces the impact of external factors, leading to a more stable and fair assessment of the LLMs.
- Extensive experiments conducted with various LLMs validate the effectiveness and generalization of `T-Eval`, providing valuable insights into bottlenecks of current LLMs, and offering new perspectives in improving tool-utilization capabilities.

2 T-Eval

Benchmarking LLMs as tool agents involves multiple dimensions of evaluations of LLM abilities and suffers from the external influence of tools. Therefore, we first thoroughly investigate each critical dimension of the tool-calling process (§2.1), and then establish tailored evaluation protocols for each dimension (§2.2), named `T-Eval`, to enable a detailed evaluation of tool utilization capability. To guarantee the high quality of the golden solution paths and tool-calling responses, we adopt a human-in-the-loop data generation pipeline (§2.3), ensuring the stability and longevity of `T-Eval`. Lastly, we provide a statistical overview of `T-Eval` (§2.4).

2.1 Evaluation Decomposition

Tool utilization with large language models (LLMs) encompasses a variety of scenarios, touching upon multiple dimensions of capabilities. To better understand the whole process, we first deconstruct the tool-calling process into several key aspects, as depicted in Fig. 1.

First, solving complex real-world problems frequently requires a multi-step approach to tool calling. For example, to know the weather from a month ago, an LLM must first confirm the current date before it can query a weather tool. This requires a robust **planning** ability (Fig. 1(a)) to develop a strategy for tool calling that guides subsequent actions. Moreover, the contexts in which tools are utilized can be intricate, involving tool descriptions, documentation, user queries, previous interactions, and prior observations. Strong **reasoning** abilities (Fig. 1(b)) are essential for LLMs to understand these contexts and tools, generating logical thoughts for the next steps. After generating a thought, selecting the appropriate tools from a given list is crucial, demanding effective **retrieval** skills (Fig. 1(c)). Additionally, integrating the correct parameters requires the **understanding** ability (Fig. 1(d)) to interpret tool documentation and corresponding thoughts. Finally, executing the tool-calling action mandates adept **instruction following** skills (Fig. 1(e)) to formulate precise requests for the relevant APIs. Each tool call ex-

ecuted by LLM must be evaluated to ensure the response meets the intended objective, especially when tools might be unavailable or not perform as anticipated. This crucial evaluation, named the **review** ability (Fig. 1(f)), involves examining tool responses and ascertaining if adequate information has been obtained to resolve the query.

In summary, thorough analyses of each dimension are vital for a comprehensive evaluation of tool-utilization capabilities. Therefore, we introduce `T-Eval`, a framework that decomposes the multi-step tool-calling process into fundamental abilities, evaluating them individually for a more nuanced understanding of tool utilization.

2.2 Fine-Grained Evaluation Protocol

`T-Eval` takes all the ability dimensions as mentioned above (PLAN, REASON, RETRIEVE, UNDERSTAND, INSTRUCT, and REVIEW) into consideration, measuring not only the overall performance of tool-utilization but also detailed scores in each dimension.

2.2.1 Definition

To formalize, this paper considers a piece of query data as a tuple (T, q), where $T = [tool_1, ..., tool_k]$ is the tool list and q is the query. For each query data piece (T, q), we define the solution path $S = [(t_i, a_i, o_i, r_i)]_1^n$ for the query q as a sequence of thought(t)-action(a)-observation(o)-review(r) pair along with the final answer A, where t_i, a_i, o_i, r_i denotes the thought, the tool-calling action, the observation (*i.e.,* the tool response), and the review on the response at step i. Moreover, an action is regarded as a pair $(tool, args)$, where $tool$ is the tool name and $args$ is the parameters to call the tool. Besides the solution path, a plan for a query data piece is defined as a sequence $P = [a_1, ..., a_n]$ donating the proposed action sequence to call at each step.

2.2.2 Single-Index Evaluation

We create individual metrics to quantitatively analyze LLM abilities for each dimension of tool utilization. Here we describe the measurement of each dimension, and the detailed metric function can be found in Appendix C.

- **PLAN**: Given a tool list T and query q, the LLM is asked to generate a proposed tool-calling action sequence $P^{pred} = [a_1^{pred}, a_2^{pred}, ..., a_n^{pred}]$, where a_i^{pred} is the predicted LLM action at step i. The planning evaluator then compares P^{pred} with the golden answer P^{gt} by matching actions in both sequences using Sentence-BERT (Reimers and Gurevych, 2019) for similarity calculation and Hopcroft-Karp matching (Hopcroft and Karp, 1973) for maximal similarity pairing. The planning score is measured by the length of the longest-ordered action sequence in the similarity pairing.

- **REASON**: Given a tool list T, query q, and a prefix of the solution path, the reasoning evaluator asks the LLM to generate the next thought t_{i+1}^{pred}. The similarity between t_{i+1}^{pred} and the golden answer t_{i+1}^{gt} is then measured.

- **RETRIEVE**: Given a tool list T, query q, and a prefix of the solution path, the retrieval evaluator asks the LLM to choose the next tool $tool^{pred}$ to call, comparing it with the golden answer $tool^{gt}$.

- **UNDERSTAND**: Given a tool list T, query q, the understanding evaluator asks the LLM to generate appropriate parameters $args^{pred}$ for the next step and then compares them with the golden answer $args^{gt}$ for similarity.

- **INSTRUCT**: Given a thought t_i with the desired tool and parameters, the LLM is tasked with generating a tool-calling request in a specified format. The evaluator then calculates the accuracy of the tool name and parameter values in this format.

- **REVIEW**: Given a thought t_i and a tool response o_i, the LLM is tasked with judging whether the tool response successfully achieves the goal mentioned in the thoughts. It must also determine the type of errors if the goal is not achieved. This evaluation is conducted as a multiple-choice problem with five options: *Success*, *Internal Error*, *Input Error*, *Irrelevant Response*, and *Unable to Accomplish*.

2.2.3 End-to-End Evaluation

End-to-end evaluation requires LLMs to generate the whole solution path S^{pred} as well as the final answer A^{pred} given one query data piece. We adopt the win rate proposed in ToolBench (Qin et al., 2023b) to gauge the overall performance, which evaluates LLM abilities by comparing their

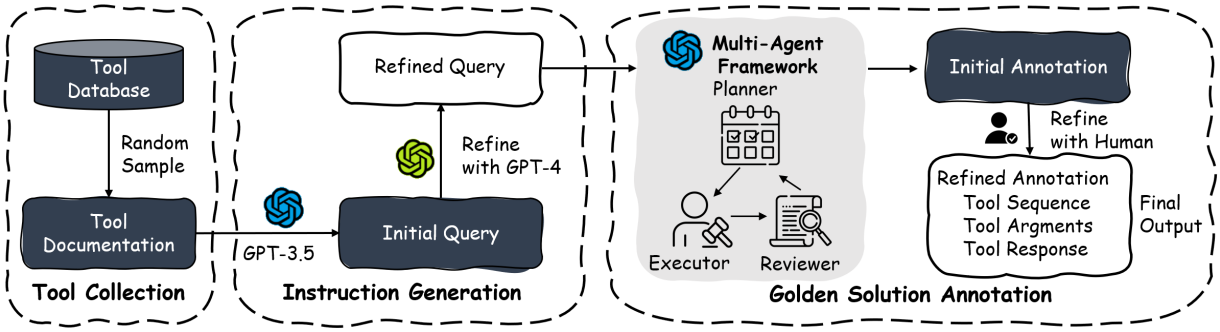


Figure 2: **Overview of the dataset construction process.** By randomly sampling tools from the tool database, we prompt GPT-3.5 to generate initial queries and further refine them with GPT-4. After that, we develop a multi-agent framework to resolve queries with the provided tools, collecting both solution paths and tool responses. Finally, human experts are employed to verify the annotations and pick high-quality samples.

response quality against that of GPT-3.5. The result shows that our single-index evaluation is consistent with the overall performance.

2.3 Dataset Construction

The construction of `T-Eval` consists of three main phases: *tool collection*, *instruction generation*, and *golden solution annotation*. The overview of the construction is shown in Fig. 2.

2.3.1 Tool Collection

The collection quality of tools has a direct impact on instruction generation and tool utilization evaluation. We follow two principles during the collection process:

• **High Availability and Usage Rate**. Considering that `T-Eval` is expected to cover most daily and practical use cases, we carefully select $1 \sim 2$ tools for each specific domain, including Research, Travel, Entertainment, Web, Life, and Financials, resulting in 15 tools as our basic tool set.

• **Complete Documentation**. Despite the numerous tools collected in ToolBench (Qin et al., 2023b) from RapidAPI, the documentation quality is not guaranteed. To reduce the failure of tool-calling cases caused by inadequate tool descriptions, which focus the evaluation attention on pure LLM abilities, we manually generate high-quality and detailed tool documentation for each tool.

2.3.2 Instruction Generation

The testing instructions determine the practicality and difficulty of our evaluation. To guarantee the diversity of the queries, we uniformly sample $2 \sim 3$ tools each time and prompt GPT-3.5 to generate N instructions $Q = \{q_1, ..., q_N\}$ that need these tools. Concretely, the prompt consists of three parts: (1) the instruction that requests LLM to generate corresponding queries, (2) detailed tool documentation, and (3) few-shot examples. We randomly shuffle the tool documentation list and select different few-shot examples each time, so that the LLM can pay different attention to the text thereby encouraging the model to create wide-ranging instructions. After that, the stronger GPT-4 is utilized to revise and refine the generated instructions, aiming to further enhance the feasibility and diversity. The detailed prompts are listed in the Appendix B.

2.3.3 Golden Solution Annotation

Annotating the solution path manually to various queries is labor-intensive and unable to scale up the dataset quickly. To overcome this problem, we leverage a novel multi-agent paradigm plugging with simple human verification to resolve the complicated and massive solution annotations. Specifically, instead of instantiating only one LLM to handle the whole solution annotation path, we explicitly disentangle the annotation task into three different functionalities, including planner, executor, and reviewer: the planner decides what should be done in the next step; the executor is responsible for generating the exact tool name as well as its parameters and executing the tool to obtain the response. The reviewer is assigned to revise the response from the tool and judge if the task is finished given the external feedback.

Thanks to the decomposition of functionalities, each agent can accomplish its duty without switching the role required by each step, therefore, significantly reducing the error generation during the annotation process compared to conventional CoT (Wei et al., 2022) or ReAct (Yao et al., 2022) approaches. To guarantee the quality of the calling process, we adopt GPT-3.5 to accomplish the whole machine annotation phase. In the end, human annotators are employed to review the solu-

tion chain and manually filter invalid instruction-solution pairs.

2.3.4 Inclusive Difficulty Evaluation

During the evaluation, we empirically found that a few LLMs, especially small-scale ones, demonstrate poor ability in instruction following, therefore yielding responses that are unparseable with the pre-agreed format. Due to the large amount of parse failures on the response, the evaluation score can get distorted, losing the authenticity to reflect the real ability of the model.

To address the issue, we carefully designed the instruction task prompt and the evaluation granularity with both easy and difficult levels, providing inclusive evaluation on most language models. Specifically, the easy level adheres to a simple string format and focuses more on the semantic quality of the text, while the difficult level adopts JSON format, which is more commonly used in products (*e.g.*, the JSON mode[1] in GPT-4 (OpenAI, 2023)), and conducts a more strict, fine-grained evaluation of the response content, *e.g.*, exact match on tool name and parameters.

2.4 Dataset Summary

To this end, we generate 1,500 initial instruction-solution pairs and pick 553 after two-round human verifications. We extract the desired information required by each evaluation protocol to construct the respective INSTRUCT, RETRIEVE, PLAN, REASON, UNDERSTAND, REVIEW subsets for `T-Eval` benchmark, resulting in 23,305 test cases in total (Refer to Appendix A.1 for more detailed statistics of `T-Eval`).

3 Experiments

3.1 Experimental Setup

We evaluate both API-based commercial and open-source LLMs on `T-Eval`, with a total number of 20 models, aiming to provide a comprehensive benchmark for current large language models.

(1) For API-based LLMs, we select three representative models: GPT-3.5 and GPT-4 from OpenAI, and Claude2 from Anthropic. [2]

(2) For open-source LLMs, we choose a wide spectrum of models, including LLaMA2 (Touvron et al., 2023), CodeLLaMA (Roziere et al., 2023), QWen (Bai et al., 2023), InternLM (Team, 2023), Baichuan2 (Yang et al., 2023), WizardLM (Xu et al., 2023b), Vicuna (Chiang et al., 2023), AgentLM (Zeng et al., 2023), Mistral (Jiang et al., 2023) and ChatGLM3 (Zeng et al., 2022).

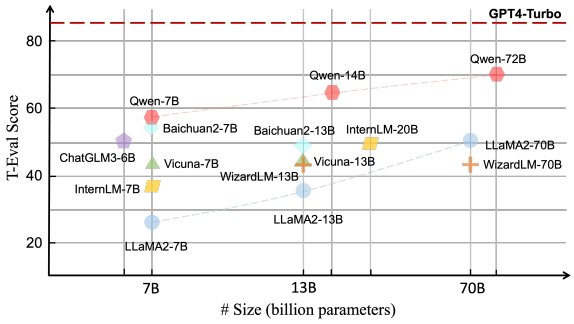


Figure 3: **`T-Eval` score v.s the size of models**. Both LLaMA2 and Qwen strengthen their tool utilization abilities as the models scale up. However, there still exists a clear performance gap between open-source models and GPT-4.

3.2 Main Results

The detailed experimental results are shown in Tab. 1. In this section, We aim to answer three research questions below.

Q1: Which Model is Better at Tool Utilization? The results in Tab. 1 show that GPT-4 achieves the highest score, with an overall score of 86.4, setting the pilot of the well-instructed and skillful tool-utilization LLMs. Apart from GPT-4, API-based commercial LLMs, including GPT-3.5 and Claude2, get competitive scores on both string and JSON formats, indicating their strong abilities in acting as tool agents.

As for open-source models, we evaluate models with three different scales: around 7B, 13B, and 70B. It can be concluded from Fig. 3 that the performance of the model monotonically increases as the model scale increases. Among them, Qwen-7B gets the best of two worlds in terms of the model sizes and evaluation scores. With 7 billion parameters, Qwen-7B exhibits a competitive ability to understand complicated instructions and reply in a strict format in JSON. When scaled to 72B, the overall score of Qwen rises to 71.4%, significantly reducing the gap between open-source and API-based models. We attribute this to the training on a human-in-loop self-instruction dataset, which encompasses high-quality format-specific instruc-

[1] https://openai.com/blog/new-models-and-developer-products-announced-at-devday

[2] Our experiments are conducted between 12/01/2023 and 12/10/2023. The version for GPT-4 is `gpt-4-1106-preview`, for GPT-3.5 is `gpt-3.5-turbo-16k`, and for Claude2 is `claude-2.1`.

Table 1: **Main Results of T-Eval.** Overall stands for the score calculated from an average of metrics on all subsets. (**bold** denotes the best score among all models, and underline denotes the best score under the same model scale.)

Model	INSTRUCT		PLAN		REASON		RETRIEVE		UNDERSTAND		REVIEW	Overall
	String	JSON	String	JSON	String	JSON	String	JSON	String	JSON	Choice	
API-Based												
Claude2	**97.7**	97.8	87.1	84.9	62.9	62.8	76.5	78.2	74.9	82.0	70.4	78.8
GPT-3.5	94.1	**99.1**	86.6	86.6	65.2	**70.3**	**98.3**	86.2	82.9	88.1	75.6	84.0
GPT-4	96.7	95.9	**88.9**	**86.7**	**65.6**	65.1	91.3	**86.6**	**83.2**	**88.3**	**94.5**	**86.4**
Open-Sourced												
LLaMA2-7B	68.7	0.2	47.0	9.1	37.1	7.1	30.3	3.5	36.8	12.1	38.6	27.4
CodeLLaMA-7B	**96.0**	0.9	61.4	44.3	28.7	0.9	3.6	1.2	25.4	1.4	40.0	28.6
AgentLM-7B	80.8	13.1	53.1	15.9	50.1	17.5	70.2	13.8	66.4	26.1	44.8	41.4
Vicuna-7B	65.3	30.8	13.4	47.8	47.6	49.9	12.4	32.6	66.8	54.2	58.5	44.8
InternLM-7B	48.4	29.9	67.7	43.1	48.8	25.0	72.1	22.2	70.4	30.2	46.2	45.8
ChatGLM3-6B	63.3	80.8	46.9	38.5	48.2	24.1	66.5	24.0	79.9	35.6	54.8	51.4
Mistral-7B	59.7	63.6	77.2	64.9	63.0	15.3	92.6	11.0	79.8	18.1	63.2	56.0
Baichuan2-7B	68.0	78.0	65.6	39.0	51.3	31.3	73.7	28.5	80.1	39.2	61.4	56.5
Qwen-7B	28.7	94.2	66.2	63.1	56.4	34.1	89.0	35.3	77.7	46.1	61.6	59.5
LLaMA2-13B	66.7	0.0	48.7	65.1	42.4	10.5	42.7	6.6	45.7	13.1	53.0	37.3
Vicuna-13B	67.0	30.8	25.8	54.0	56.3	49.1	19.8	20.9	73.0	58.8	60.8	48.1
WizardLM-13B	14.1	65.5	77.7	40.8	36.0	25.2	68.5	22.2	64.0	31.6	71.5	49.0
Baichuan2-13B	8.0	51.7	69.5	52.1	56.6	27.1	84.5	26.9	80.5	31.5	57.3	50.3
Qwen-14B	49.7	97.6	79.6	69.7	58.6	46.1	**95.9**	55.3	65.0	**64.3**	56.9	66.3
InternLM-20B	43.6	31.9	83.6	16.4	63.0	28.1	81.5	40.6	80.1	43.5	64.7	53.5
WizardLM-70B	9.6	31.7	81.5	42.7	38.4	47.0	38.3	56.2	66.1	61.1	28.7	44.2
LLaMA2-70B	84.5	73.4	58.0	63.1	44.7	17.5	62.0	17.1	67.3	22.3	62.8	53.0
Qwen-72B	27.8	**98.3**	**85.1**	**73.4**	**63.5**	**55.4**	76.8	**65.0**	**84.5**	66.1	**80.3**	**71.4**

tions generated by Qwen team (Bai et al., 2023).

Q2: How Far Are We from Skillful Tool Agents? By explicitly disentangling the evaluation through model abilities, we can gain a deeper understanding of the pros and cons of current LLMs, providing new perspectives in developing better tool agents.

First, open-source LLMs lack the instruction-following ability to respond with specific formats, which is the very first step to constructing well-regulated and high-usable tool agents. Without a legal response format, the system can not successfully extract the information generated by LLMs by a constant protocol, not to mention the correct tool executions. However, only a small amount of models achieve both high scores on INSTRUCT subset under string and JSON format. Besides, there are still large performance gaps between string and JSON evaluation protocols on other subsets. Considering that the *understand* ability of the Qwen-72B is comparable with GPT-4 evaluated in string format (84.5 vs 83.2), its JSON format result is more than 20 points lower, pinpointing the necessity to enhance the ability of open-source LLMs solving problems with specific formats.

Second, tool retrieval presents a relatively challenging task for most LLMs. Even the largest open-source model, Qwen-72B only reaches 65.0%, which is more than 20 points lower than GPT-3.5.

Lastly, compared to planning, LLMs are more likely to perform worse in reviewing the status of API responses (*i.e.,* review in T-Eval), which is a core capability to interact with the dynamic environment when acting as an agent. Most models only reach 50%∼60%, compared to 95% achieved by GPT-4, indicating that more attention should be paid to the review ability of current LLMs.

Q3: What Makes for Good Training Data for Tool Utilization? Supervised finetuning is an efficient and necessary practice in empowering LLMs with certain abilities on downstream tasks. Recent

research (Ouyang et al., 2022; Touvron et al., 2023) finds that high-quality dialogue-style instruction data is the key to well-instructed LLMs. However, the analysis of what makes for good training data for tool utilization is under-explored. We identify two types of training data: *(1) general instruction following data*, and *(2) task-specific tuning data*.

As for high-quality general instructions, Vicuna adopts user-shared conversations collected from ShareGPT[3], and WizardLM uses complex and diverse instructions by evolving existing data in depth and width. Both of them are trained starting from LLaMA2, providing natural ablation on the effectiveness of high-quality (diverse and complex) instructions to the tool learning ability of LLM. When the model scale is small (7B), these data types enhance the model considerably. However, the increments diminish (even worse) as the model scales up (see WizardLM-70B). This further indicates that simply scaling the model scale does not always bring improvements, proper training data also matter in the scaling law.

In terms of task-specific tuning data, we select two typical types of corpus: code and agent, corresponding to CodeLLaMA and AgentLM, respectively. Compared to CodeLLaMA, which uses code data, AgentLM obtains better scores, showcasing that agent-related data may bring more benefits to tool learning. However, neither CodeLLaMA nor AgentLM appears to have significant advantages to Vicuna, indicating the necessity of high-quality instruction following data for tool utilization.

4 Discussion

4.1 Format Following v.s Problem Solving

Format following is an essential ability of LLMs, *i.e.,* reply with a specific format when executing certain tasks. There are massive efforts devoted to enhancing this ability when using LLM as agents (Zhou et al., 2023; Xu et al., 2023b). In our experiments, we find that this ability may need to be acquired jointly with problem-solving. For instance, both ChatGLM3-6B and Baichuan2-7B obtain roughly 80% with the JSON evaluation protocol on INSTRUCT subset, which reveals them holding a strong ability in JSON format rendering. However, they struggle to generate valid JSON format responses on PLAN and REASON subsets, *i.e.,* the divergences of JSON and string scores are quite large on these subsets. Such a phenomenon suggests that the ability to output specific formats, *i.e.,* JSON, does not guarantee the ability to resolve all the problems with this format. This further indicates that one should integrate the requested format into the tasks and train them jointly so that the model can understand and behave well under certain protocols.

4.2 Inclusive Evaluation Protocol

From Tab. 1, it can be observed that quite a few amount of open-source LLMs struggle on the JSON evaluated protocols, especially on the REASON, RETRIEVE and UNDERSTAND subsets. Although the JSON format evaluation best approximates the real use cases of tool agents, it fails to provide hierarchy discriminations across various models when they are not adept at specific instructions requested by the task. For instance, Baichuan2-13B exhibits poor abilities in JSON format instruction (only 51.7 on INSTRUCT, JSON), which leads to low scores under JSON evaluations in PLAN subset. However, it has little relationship with the basic ability of model planning, since it achieves 69.1% when evaluated with string format, which is 17.4 points higher than its JSON format (52.1%). This validates the necessity to provide continual difficulty level evaluation protocols (Schaeffer et al., 2023), otherwise, one can get little understanding of the detailed abilities of LLMs on this benchmark but a low score simply due to the incorrect format, especially for weak models. In `T-Eval`, by seamlessly converting the strict format matching into semantic sentence analysis, our inclusive evaluation protocol gets rid of the inflexible measurement, unearthing the inner capability of the model.

4.3 Comparison to Other Benchmarks

We compare our fine-grained evaluation protocols with existing tool evaluation approaches and investigate if they show the same trends as ours. We adopt the win rate proposed in ToolBench (Qin et al., 2023b) as the representative holistic evaluation method, and evaluate several open-source models by comparing the response quality with GPT-3.5-turbo, judged by GPT-4. The results are shown in Fig. 4. We can find that the holistic evaluation reveals similar trends in these models with that in `T-Eval`, which validates the reasonability and generalization of our benchmark. When taking a close look at the results, we can observe that Qwen-7B achieves a 52% win rate over GPT-3.5-turbo response. However, there still exists a gap between

[3] https://sharegpt.com/

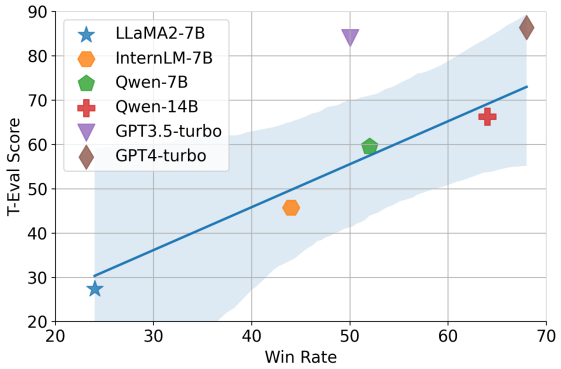


Figure 4: T-Eval average score v.s Win Rate proposed in ToolBench (Qin et al., 2023b) on several representative LLMs. T-Eval score (objective) demonstrates similar trends with Win Rate (judged by GPT-4).

Qwen-7B and GPT-3.5 in various ability domains under human judgment (Zheng et al., 2023), which implies that holistic evaluation is sometimes inaccurate. Our evaluation protocol not only reflects such divergence clearly, but also showcases the details of abilities in tool utilization, suggesting that T-Eval to be a more rational and comprehensive evaluation benchmark for tool utilization.

5 Related Work

Augmenting LLMs with Tools There are two paradigms to empower LLMs with external tools, and the first one is regarding external tools as specific tokens and fine-tuning parts or full of the model (Schick et al., 2023; Parisi et al., 2022; Lewis et al., 2020; Hao et al., 2023). However, these methods need a large amount of tool-relevant data and struggle to adapt newly appeared tools. Recently, the strong in-context learning ability (Brown et al., 2020) promotes researchers to focus more on the second paradigm, which is augmenting LLMs with tools by giving in-context tool descriptions and demonstrations (Hsieh et al., 2023; Mialon et al., 2023; Ruan et al., 2023; Patil et al., 2023). This paradigm has achieved great tool-calling potentiality and resulted in successful applications such as ChatGPT plugins. T-Eval focus on this paradigm and evaluate scores of various foundation models.

Evaluating LLMs LLM evaluation is essential to ensure that LLM can be effective in understanding and generating human-preferred text and reliable for deployment in real-world applications (Guo et al., 2023; Chang et al., 2023). Many benchmarks have been established to evaluate base abilities on question-answering tasks (Rajpurkar et al., 2016; Clark et al., 2018; Glockner et al., 2018), natural language understanding tasks (Wang et al., 2018, 2019; Hendrycks et al., 2020), and commonsense reasoning tasks (Lu et al., 2022). Recently, LLM evaluation has extended towards specific directions like code generation (Chen et al., 2021; Austin et al., 2021; Du et al., 2023) and hallucination (Li et al., 2023a; Chen et al., 2023). Some benchmarks also test the performance of LLM-based agents in a wide range of scenarios (Liu et al., 2023a; Wang et al., 2022).

Several benchmarks exist for evaluating tool utilization, focusing primarily on aspects of response comparison (e.g., ToolQA (Zhuang et al., 2023)), tool call accuracy (e.g., Gorilla (Patil et al., 2023)), or a combination of both (e.g., API-Bank (Li et al., 2023b)). ToolBench (Qin et al., 2023b) introduces a novel approach by employing an LLM as a judger to assess the overall solution path. Furthermore, the study in (Qin et al., 2023a) investigates the performance improvement attributable to tool utilization. Different from above, T-Eval emerges as the first benchmark dedicated to the fine-grained evaluation of tool utilization capabilities.

Prompting LLMs as Agents Prompting enhances the reasoning capabilities of LLMs by providing instructions or examples. Techniques such as the Chain of Thought (CoT) and Tree of Thought (ToT) (Wei et al., 2022; Yao et al., 2023) encourage LLMs to engage in comprehensive thinking for more accurate reasoning. Advanced systems like ReAct, ReWOO, SwiftSage, DyLAN, and DP-LLM (Yao et al., 2022; Xu et al., 2023a; Lin et al., 2023; Liu et al., 2023b; Dagan et al., 2023) further develop LLM agents. These systems use advanced prompting methods to guide LLMs, unleashing the potential of models.

6 Conclusion

In this paper, we propose T-Eval, a comprehensive and fine-grained tool utilization evaluation benchmark for LLMs. T-Eval explicitly disentangles the tool utilization tasks along the model ability, with dedicated evaluation protocols designed for respective tasks, unearthing the real ability of the evaluated models. Such a step-wise evaluation delivers a thorough analysis and pinpoints the main bottlenecks of current LLMs in tool learning, providing valuable insights into further development of tool agents.

7 Limitations

Despite T-Eval mainly utilizing the advanced GPT model for dataset generation, it still requires human efforts to verify the correctness of the generated content. It is essential to acknowledge that the generated content may not be well aligned with human preference. Therefore, it would be hard to scale up the construction process to a larger scale. We aim to address this problem in our future work.

8 Ethical Considerations

We used publicly available reference documents/APIs for our benchmarks, effectively circumventing any possible harm toward individuals or groups. The generated data by LLMs were carefully selected and processed by humans to secure privacy and confidentiality. No personal identification information was involved, and all data were made anonymous before any analysis was conducted. Besides, We use ChatGPT and Grammarly to polish the writing.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2023. Benchmarking large language models in retrieval-augmented generation. *arXiv preprint arXiv:2309.01431*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *https://vicuna.lmsys.org*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *arXiv preprint arXiv:2308.01861*.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking nli systems with sentences that require simple lexical inferences. *arXiv preprint arXiv:1805.02266*.

Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, Deyi Xiong, et al. 2023. Evaluating large language models: A comprehensive survey. *arXiv preprint arXiv:2310.19736*.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

John E Hopcroft and Richard M Karp. 1973. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231.

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

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

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023a. Halueval: A large-scale hallucination evaluation benchmark for large language models. *arXiv e-prints*, pages arXiv–2305.

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

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *arXiv preprint arXiv:2305.17390*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023a. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023b. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*.

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

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

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

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

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, and Yusheng Su et al. 2023a. Tool learning with foundation models.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

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

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

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

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? *arXiv preprint arXiv:2304.15004*.

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

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities.

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

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023a. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

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

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

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

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

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

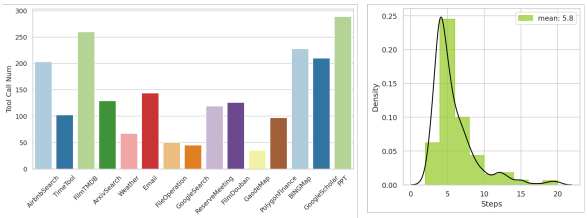
Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*.

A T-Eval Benchmark Details

A.1 Dataset Statistics

T-Eval originates from 533 high-quality query-solution annotation pairs, consisting of 23,305 test cases in total, ranging from INSTRUCT, PLAN, REASON, RETRIEVE, UNDERSTAND and REVIEW subsets. Detailed statistics of each subset are shown in Tab. 2. We also visualize the distribution of tool calling steps in the whole annotation paths in Fig. 5. T-Eval covers all tool sets and yields 5.8 average calling steps for each query, validating the generalization and discrimination for tool utilization evaluation.



(a) (b)

Figure 5: (a) Tool calling categorical distribution and (b) tool calling step distribution accumulated in the whole annotation paths in T-Eval.

Table 2: The statistics of the evaluation datasets in T-Eval.

Dataset	Test Cases
INSTRUCT	2660
RETRIEVE	6426
PLAN	553
REASON	6426
REVIEW	487
UNDERSTAND	6753
Total	23305

B Implementation Details

Experimental Details. To evaluate the pure ability of the single model, we adopt ReAct (Yao et al., 2022) as the basic agent paradigm for end-to-end evaluation and limit the maximum action step to

20 to ensure the efficient and accurate question-solving ability of LLM. As for the single-index evaluation, we prompt with a multi-turn conversation style to the LLM and gauge the response. If not specified, we choose the 'chat'/'instruct' version of open-sourced models for evaluation.

B.1 Prompts Demonstration

Please refer to the respective prompt block for a detailed demonstration.

B.1.1 Query Generation

The corresponding prompt is presented in Fig. 6.

B.1.2 Query Refinement

The corresponding prompt is presented in Fig. 7.

B.1.3 Multi-Agent Annotation Prompt

The corresponding prompt is presented in Fig. 8.

B.2 Dataset Demonstration

Please refer to the respective prompt block for each detailed dataset demonstration.

B.2.1 INSTRUCT

The corresponding prompt is presented in Fig. 9.

B.2.2 PLAN

The corresponding prompt is presented in Fig. 10.

B.2.3 REASON

The corresponding prompt is presented in Fig. 11.

B.2.4 RETRIEVE

The corresponding prompt is presented in Fig. 12.

B.2.5 UNDERSTAND

The corresponding prompt is presented in Fig. 13.

B.2.6 REVIEW

The corresponding prompt is presented in Fig. 14.

C Detailed Evaluation Metrics

`T-Eval` decomposes tool utilization capability into six ability dimensions: INSTRUCT, PLAN, REASON, RETRIEVE, UNDERSTAND and REVIEW, we carefully designed evaluators and metrics for all dimensions with two difficulty formats: JSON and string. The JSON format asks the LLM to generate standard JSON format responses, while the string format allows the LLM to answer in a relatively loose format.

Let us recap the formalization of each tool-calling component first: A piece of query data is considered as a tuple (T, q), where $T = [tool_1, ..., tool_k]$ is the tool list with k tools and q is the query. For each query data piece (T, q), the solution path $S = [(t_i, a_i, o_i, r_i)]_1^n$ is defined as a sequence of thought(t)-action(a)-observation(o)-review(r) pair along with the final answer A, where t_i, a_i, o_i, r_i denotes the thought, the tool-calling action, the observation (i.e. the tool response), and the review on the response at step i, respectively. Moreover, an action a is regarded as a pair $(tool, args)$, where $tool$ is the tool name and $args$ is the parameters to call the tool. Besides the solution path, a plan for a query data piece is defined as a sequence $P = [a_1, ..., a_n]$ donating the proposed actions to call at each step.

C.1 INSTRUCT

The LLM is required to generate a tool-calling request using a specified template, based on the provided tool name and parameters. This request must adhere to a predetermined format, either in JSON or string, with varying template structures. Initially, the evaluator determines if the request meets the format requirements. A *passing score* of 0.5 is awarded for successfully meeting these format standards. Once the format check is passed, the request is further evaluated for parameter accuracy, with a *parameter score* assigned. This score is calculated as 0.5 multiplied by the percentage of correctly matched parameters. The final score is the sum of the *passing score* and the *parameter score*.

C.2 PLAN

The LLM is tasked with generating a plan using a provided list of tools to solve a query. To evaluate the similarity between the predicted plan from the LLM $P^{pred} = [a_1^{pred}, a_2^{pred}, ..., a_{n^{pred}}^{pred}]$ and the gold answer $P^{gt} = [a_1^{gt}, a_2^{gt}, ..., a_{n^{gt}}^{gt}]$ from human annotators, the planning evaluator begins by computing a similarity matrix S. This matrix represents the similarity scores for all action pairs $(a_i = (tool_i, args_i), a_j = (tool_j, args_j))$ between the prediction and the golden answer:

$$S_{i,j} = \beta\sigma(tool_i, tool_j) + (1-\beta)\sigma(args_i, args_j).$$

In this approach, σ is the similarity function between two sentences. We employ Sentence-BERT (Reimers and Gurevych, 2019) as σ, which involves embedding the two sentences and then calculating the cosine similarity between these embeddings as

the similarity score. The underlying BERT model used is `all-mpnet-base-v2`.[4] Furthermore, β is a hyperparameter that determines the relative importance of the tool name in comparison to the tool parameters in similarity calculation. In our implementation, we set $\beta = 0.75$.

After getting the similarity matrix S, a bipartite graph is built where one part is the set of predicted actions and another part is the set of golden answer actions. Two actions are linked if their similarity is greater than a predefined threshold, set at 0.7 in our implementation. We then employ the Hopcroft-Karp matching algorithm (Hopcroft and Karp, 1973) to compute a max-weighted match from this graph. Subsequently, the Longest Increasing Subsequence (LIS) algorithm is used to determine the longest-ordered action sequence within this max-weighted match. Denoting the length of this sequence as l, we calculate the precision and recall as $p = l/n^{pred}$ and $r = l/n^{gt}$, respectively. The plan score is thus defined as:

$$\text{plan score} = \frac{2pr}{p+r}.$$

Regarding the input formats, in the JSON format, the LLM is tasked with generating a list of actions, with each action represented as a dictionary comprising the tool name and its corresponding parameters. Conversely, in the string format, the LLM articulates each action in a separate line.

C.3 Reason

Given a tool list T, query q, and a prefix of the solution path, the LLM is asked to generate the next thought t_{i+1}^{pred}. The similarity between t_{i+1}^{pred} and the golden answer t_{i+1}^{gt} is then measured, and the similarity is calculated by Sentence-BERT (the same as the planning evaluator). In the JSON format, LLMs need to generate a dictionary containing the next thought, as well as the next tool name and the corresponding parameters, which means we evaluate Reason, Retrieve, Understand using the same LLM output. Regarding the string format, LLMs only need to generate the next thought in a single line.

C.4 Retrieve

Given a tool list T, query q, and a prefix of the solution path, the LLM is asked to generate the next tool name $tool_{i+1}^{pred}$ to call, and then the evaluator judges whether the name is the same as the golden answer $tool_{i+1}^{gt}$, achieving score 1 if the same and 0 otherwise. As mentioned in Sec. C.3, the LLM needs to generate a dictionary containing the next thought, as well as the next tool name and its corresponding parameters in the JSON format, while in the string format, the golden answer's thought is given, and the LLM only needs to generate the next tool name in a single line.

C.5 Understand

Given a tool list T, query q, and a prefix of the solution path, the LLM is tasked with generating the parameters $args_{i+1}^{pred}$ to call the next tool, and the score is the similarity between $args_{i+1}^{pred}$ and $args_{i+1}^{gt}$ calculated by Sentence-BERT (the same as planning evaluator). As mentioned in Sec. C.3, the LLM needs to generate a dictionary containing the next thought, as well as the next tool name and its corresponding parameters in the JSON format, while in the string format, the golden answer's thought and tool name are given, and the LLM needs to generate the parameters in a single line.

C.6 Review

Given a thought t_i and a tool response o_i, the LLM is required to evaluate the tool response. It must select one of the following categories to classify the response: *Success*, *Internal Error*, *Input Error*, *Irrelevant Response*, or *Unable to Accomplish*. The evaluation is scored as 1 for a correct classification and 0 if the classification is incorrect.

D API Documentation

We manually curate extensive API documentation for each tool, following the annotation format defined by OpenAI. Compared to official RapidAPI documentation, our descriptions are more extensive and detailed, which facilitates the understanding of various tools and circumvents the failure cases due to the incomplete API documentation provided in the benchmark. Here, we provide the API documentation for BINGMap as an example in Fig. 15.

[4] https://www.sbert.net/docs/pretrained_models.html

B.1.1 Query Generation

Prompt:

You will be given a tool list with a few API descriptions. Please carefully read the API documentation. Your task involves creating 3 varied, innovative, and concrete queries. Respond strictly with JSON. The JSON should be compatible with the TypeScript type `Response` from the following:

```
interface Response {
    // 3 generated responses based on the given tools
    cases: [
        0: {
            // list of tools selected, in the format "tool_name.api_name" e.g.,
            AirbnbSearch.search_property_by_place
            tools: list;
            // describes one specific role, it should be a common role in our daily life
            role: string;
            // describes the detailed query content
            query: string;
        };
        // rest 2 use cases
        ...
    ];
}
```

Rule:
1. API name is strictly prohibited from appearing in the generated query.
2. Each query must use all tools: {tool_names}. Query that only calls one tool will NOT be accepted.
3. The QUERY and ROLE must be totally different from your previous response.
4. The query should contain every detail, especially parameter inputs required by the APIs, so that it can be completed **without** using any external information.
5. The information provided in the query MUST truly exist, especially unique IDs required in APIs.
6. The maximum requested number of items should be limited to 3.

Figure 6: An example prompt of query generation.

B.1.2 Query Refinement

Prompt:

You will be provided with 3 queries. Please carefully review and update the query so that the query can be successfully executed. Respond strictly with JSON. The JSON should be compatible with the TypeScript type `Response` from the following: {response_format}
Rules:
1. API name (e.g., search_property_by_coordinates) is strictly prohibited from appearing in the query.
2. Provide the exact and real information in the query, Do NOT provide template information, e.g., YOUR_FILE_PATH.
3. Avoid saying 'a specific xxx', 'the first/second xxx' is preferred.
4. Fake information is strictly prohibited in the query (e.g., 1234567). You can modify part of the query so that the desired information can be obtained by other APIs to avoid generating these fake information.

Figure 7: An example prompt of query refinement.

B.1.3 Multi-Agent Annotation Prompt

System Prompt:

Answer the following questions as best you can. Specifically, you have access to the following APIs: {func_list}. Respond strictly with JSON. The JSON should be compatible with the TypeScript type `Response` from the following:

```
interface Response {
    // task id of the action
    id: int;
    // name of the action, must be function name
    name: string;
    // dependency/prerequisite of current action, list of task id
    dep: list;
    // input params required by current action.
    args: Record<string, any>;
    // the exact goal of executing this action
    goal: string;
}
```

Remember:
1. ONLY generate one action at each time.
2. If you have finished ALL tasks requested by the query, please reply: {finish_example}
Begin!

Reviewer Prompt:

You are an expert in discriminating if the task is accomplished. You will be provided with the following information:

```
Goal: the goal of the current task
Action: tool name of the current task
Response: response from the called tool
```

Respond strictly with JSON. The JSON should be compatible with the TypeScript type `Response` from the following:

```
interface Response {
    // explain why the task is accomplished/unaccomplished
    thought: string;
    // whether the task is done
    is_finished: boolean;
}
```

Begin!

Figure 8: An example prompt of Multi-Agent Annotation Prompt.

B.2.1 Dataset Demonstration – INSTRUCT

System:

You have access to the following API:

```
{
    'name': 'AirbnbSearch.search_property_by_place',
    'description': 'This function takes various parameters to search properties on Airbnb.',
    'required_parameters': [{'name': 'place', 'type': 'STRING', 'description': 'The name of the
    destination.'}],
    'optional_parameters': [],
    'return_data': [{'name': 'property', 'description': 'a list of at most 3 properties,
    containing id, name, and address.'}]
}
```

Please generate the response in the following format:

```
{
    goal: goal to call this action
    name: API name to call
    args: JSON format API args in ONLY one line
}
```

User:

Call the function AirbnbSearch.search_property_by_place with the parameter as follows: 'place' is 'Berlin';

Figure 9: An example prompt in the INSTRUCT dataset.

B.2.2 Dataset Demonstration – PLAN

System:

You have access to the following API:

```
{API_docs}
```

Please generate a plan for answering the user's questions, which should be a list of actions with the following format:

```
[{
    // id of the action
    "id": number;
    // the name of the action
    "name": string;
    // input params required by this action
    "args": str(Record<string, any>);
}, ...
]
```

You can imagine args when you plan the action, and these instructions will be executed sequentially. For example, if you want to call api1 with arg1 and arg2, you can write the following plan:

```
[
    {
        "id": 0,
        "name": "api1",
        "args": "{'arg1': 'value1', 'arg2': 'value2', ...}",
    }, ...
]
```

The args should be a dictionary in string format. PLEASE use '' in the args dictionary and use "" in other places, DO NOT print args with value None or null.
You should only generate a list in JSON format. The list should be the full planning list without '...' DO NOT generate any text to explain the JSON.

User:

As a researcher studying sustainable energy technologies, I need to find properties in Berlin and review no more than three of these properties. Moreover, I need to find articles on Arxiv related to 'solar energy' and get the meta-information for up to three of these articles.

Figure 10: An example prompt in the PLAN dataset.

B.2.3 Dataset Demonstration – REASON

System:

You are an assistant who can utilize external tools. You can call the following tools:

```
{API_docs}
```

If you already know the answer, please call the FinishAction to provide the final response to the answer.

User:

As a researcher studying sustainable energy technologies, I need to find properties in Berlin and review no more than three of these properties. Moreover, I need to find articles on Arxiv related to 'solar energy' and get the meta-information for up to three of these articles.

User:

What is your thought at the current step?

Figure 11: An example prompt in the REASON dataset.

B.2.4 Dataset Demonstration – RETRIEVE

System:
You are an assistant who can utilize external tools. You can call the following tools:

```
{API_docs}
```

If you already know the answer, please call the FinishAction to provide the final response to the answer.
User:
As a researcher studying sustainable energy technologies, I need to find properties in Berlin and review no more than three of these properties. Moreover, I need to find articles on Arxiv related to 'solar energy' and get the meta-information for up to three of these articles.
User:
What is your thought at the current step?
Assistant:
Find properties in Berlin.
User:
What is the tool name to call at the current step?

Figure 12: An example prompt in the RETRIEVE dataset.

B.2.5 Dataset Demonstration – UNDERSTAND

System:
You are an assistant who can utilize external tools. You can call the following tools:

```
{API_docs}
```

If you already know the answer, please call the FinishAction to provide the final response to the answer.
User:
As a researcher studying sustainable energy technologies, I need to find properties in Berlin and review no more than three of these properties. Moreover, I need to find articles on Arxiv related to 'solar energy' and get the meta-information for up to three of these articles.
User:
What is your thought at the current step?
Assistant:
Find properties in Berlin.
User:
What is the tool name to call at the current step?
Assistant:
AirbnbSearch.search_property_by_place
User:
What is the value of 'place' required by the current tool?

Figure 13: An example prompt in the UNDERSTAND dataset.

B.2.6 Dataset Demonstration – REVIEW

System:

You are presented with information about a task and its corresponding responses in the following format:

```
Goal: [The intended goal or objective of the current task.]
Name: [The name of the tool used for the task.]
Args: [The arguments or parameters supplied to the tool.]
Response: [The response or output received from the tool after execution.]
```

Based on this information, your task is to evaluate whether the goal has been achieved. Select the most appropriate option from the choices below to describe the task's outcome:
A: Success – The task has been completed successfully and the goal is achieved. B: Internal error – There is a malfunction or connectivity issue within the tool itself, leading to failure. C: Input Error – The tool is mismatched to the query, or the parameters or arguments provided to the tool are incorrect or inadequate for the task. D: Irrelevant Response – The output from the tool does not align with the expected response as per the tool's description or is ambiguous. E: Unable to Accomplish – The tool's response indicates that the task is impossible to accomplish or the tool's response is empty.
Your output should follow the following format:

```
Answer: [Insert your choice here, choosing from A, B, C, D, and E. This should be a single
character.]
```

Note that the place, the date, and the id in the parameters and the response are correct; please do not judge the correctness of the parameters and the response based on the place, the date, and id.

User:

```
Goal: prints the details of the movie 'The Battle at Lake Changjin'\\
Name: FilmDouban.print\_detail\\
Args: {'film\_name': 'Inception'}\\
Response: {'text': 'Can not find the movie named Inception'}\\
```

Figure 14: An example prompt in the REVIEW dataset.

D. API Documentation – BINGMap

```
tool_description = dict(
    name='BINGMap',
    standardized_name='bing_map',
    tool_description="Plugin for lookup map information in America",
    category="Life",
    api_list = [
        dict(
            name="get_distance",
            description="Get the distance between two locations in km.",
            required_parameters=[dict(name='start', type='STRING', description='The start
            location.'),
                dict(name='end', type='STRING', description='The end location.')],
            optional_parameters=[],
            return_data=[dict(name="distance", description="the distance in km.")]
        ),
        dict(
            name="get_route",
            description="Get the route between two locations in km.",
            required_parameters=[dict(name='start', type='STRING', description='The start
            location.'),
                dict(name='end', type='STRING', description='The end location.')],
            optional_parameters=[],
            return_data=[dict(name="route", description="the route, a list of actions.")]
        ),
        dict(
            name="get_coordinates",
            description="Get the coordinates of a location.",
            required_parameters=[dict(name='location', type='STRING', description='the location
            need to get coordinates.')],
            optional_parameters=[],
            return_data=[
                dict(name="latitude", description="the latitude of the location."),
                dict(name="longitude", description="the longitude of the location.")
            ]
        ),
        dict(
            name="search_nearby",
            description = "Search for places nearby a location, within a given radius, and return
            the results into a list. Put the location name at the end of the query.",
            required_parameters=[dict(name='search_term', type='STRING', description='the place
            name')],
            optional_parameters=[
                dict(name='places', type='STRING', description='the name of the location.'),
                dict(name='latitude', type='FLOAT', description='the latitude of the location.'),
                dict(name='longitude', type='FLOAT', description='the longitude of the location.'),
                dict(name='radius', type='NUMBER', description='radius in meters.')
            ],
            return_data=[
                dict(name="places", description="the list of places, each place is a dict with
                name and address, at most 5 places.")
            ]
        ),
    ]
)
```

Figure 15: An example API document: BINGMap.