TASKBENCH: BENCHMARKING LARGE LANGUAGE MODELS FOR TASK AUTOMATION

Yongliang Shen[1*], **Kaitao Song**[2*†], **Xu Tan**[2], **Wenqi Zhang**[1],
Kan Ren[2], **Siyu Yuan**[3], **Weiming Lu**[1†], **Dongsheng Li**[2], **Yueting Zhuang**[1†]
Zhejiang University[1], Microsoft Research Asia[2], Fudan University[3]
`{syl,luwm,zhangwenqi}@zju.edu.cn, syyuan21@m.fudan.edu.cn`
`{kaitaosong, xuta, dongsli}@microsoft.com`

`https://github.com/microsoft/JARVIS`

ABSTRACT

Recently, the incredible progress of large language models (LLMs) has ignited the spark of task automation, which decomposes the complex tasks described by user instructions into sub-tasks, and invokes external tools to execute them, and plays a central role in autonomous agents. However, there lacks a systematic and standardized benchmark to foster the development of LLMs in task automation. To this end, we introduce TASKBENCH to evaluate the capability of LLMs in task automation. Specifically, task automation can be formulated into three critical stages: task decomposition, tool selection, and parameter prediction to fulfill user intent. This complexity makes data collection and evaluation more challenging compared to common NLP tasks. To generate high-quality evaluation datasets, we introduce the concept of Tool Graph to represent the decomposed tasks in user intent, and adopt a back-instruct method to simulate user instruction and annotations. Furthermore, we propose TASKEVAL to evaluate the capability of LLMs from different aspects, including task decomposition, tool selection, and parameter prediction. Experimental results demonstrate that TASKBENCH can effectively reflects the capability of LLMs in task automation. Benefiting from the mixture of automated data construction and human verification, TASKBENCH achieves a high consistency compared to the human evaluation, which can be utilized as a comprehensive and faithful benchmark for LLM-based autonomous agents [1].

1 INTRODUCTION

Due to the recent advances of large language models (LLMs) (Brown et al., 2020; Ouyang et al., 2022; Team et al., 2023; OpenAI, 2023; Touvron et al., 2023a; Anil et al., 2023), LLM-empowered autonomous agents (Gravitas, 2023; Shen et al., 2023; Nakajima, 2023; Liang et al., 2023) have unveiled remarkable potential towards artificial general intelligence and become a new rising trend in the realm of AI research. Generally, within the realm of LLM-empowered autonomous agents, task automation is considered as the most important component, which aims to leverage LLMs to autonomously analyze user instructions and accomplish their objectives. Consequently, many researchers attempt to delve deeper into LLM to enable more intelligent task automation (Hong et al., 2023; Li et al., 2023b; Park et al., 2023; Wang et al., 2023a). However, it is worth noting that a critical challenge in advancing this area is the lack of a systematic and standardized benchmark to thoroughly evaluate the capability of LLMs in automating tasks. Therefore, creating such a benchmark to facilitate research in this area has become an urgent need.

---

[*] The first two authors have equal contributions. This work was done when the first author was an intern at Microsoft Research Asia.

[†] Corresponding author.

[1] `https://github.com/microsoft/JARVIS/tree/main/taskbench`

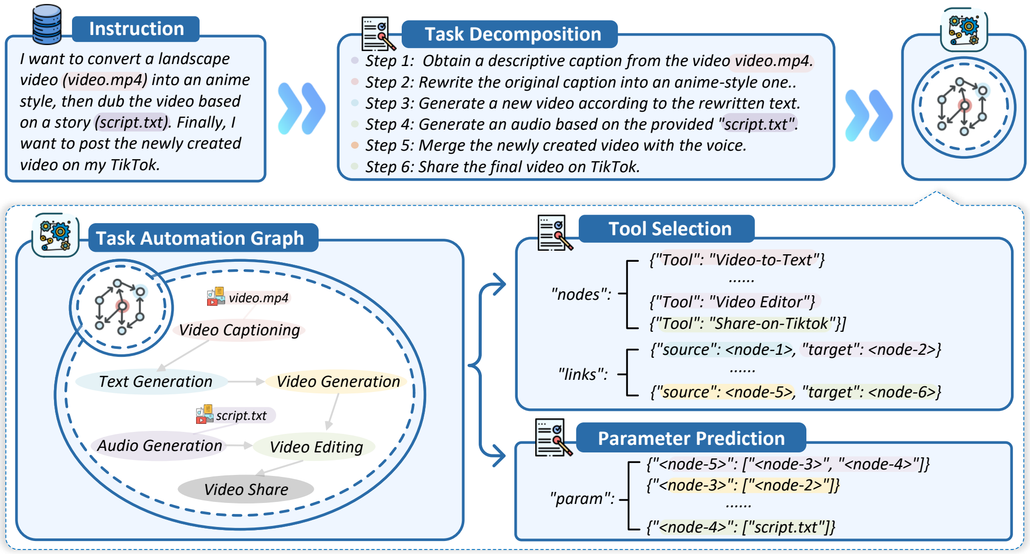

Instruction
I want to convert a landscape video (video.mp4) into an anime style, then dub the video based on a story (script.txt). Finally, I want to post the newly created video on my TikTok.
Task Decomposition
Step 1: Obtain a descriptive caption from the video video.mp4.
Step 2: Rewrite the original caption into an anime-style one..
Step 3: Generate a new video according to the rewritten text.
Step 4: Generate an audio based on the provided "script.txt".
Step 5: Merge the newly created video with the voice.
Step 6: Share the final video on TikTok.
Task Automation Graph
video.mp4
Video Captioning
Text Generation
Video Generation
script.txt
Audio Generation
Video Editing
Video Share
Tool Selection
"nodes": {"Tool": "Video-to-Text"} ...... {"Tool": "Video Editor"} {"Tool": "Share-on-Tiktok"}]
"links": {"source": <node-1>, "target": <node-2>} ...... {"source": <node-5>, "target": <node-6>}
Parameter Prediction
"param": {"<node-5>": ["<node-3>", "<node-4>"]} {"<node-3>": ["<node-2>"]} ...... {"<node-4>": ["script.txt"]}

Figure 1: LLM-based task automation. Task automation implies that LLM-based agents use task decomposition, tool selection, and parameter prediction to autonomously complete tasks. Given a user request, the large language model carries out task decomposition and predicts the tool invocation graph. The task decomposition is a series of task step descriptions, while the tool invocation graph includes the tool selection and tool parameter prediction.

Nevertheless, it is non-trivial to build such a benchmark for task automation since its setting is closer to real-world scenarios that makes the collection of evaluation data and the design of evaluation metrics more challenging than conventional NLP tasks. Figure 1 illustrates a simple example to outline the pipeline of task automation, and we can have these observations:

- In contrast to conventional NLP tasks, the procedure to fulfill task automation usually requires multiple stages (e.g., task decomposition, tool selection, and parameter prediction of tools). This indicates that we need to take all of these elements into consideration when building benchmark datasets and evaluation metrics;
- In particular, the implementation of task automation necessitates a broader consideration of real-world scenarios. As a result, the user instruction could be complex. For example, it could be composed of multiple sub-tasks with complex task dependencies. And sometimes, its task scope could demand advanced functions beyond language and thus require the utilization of external tools. These issues also verify the difficulty in constructing the benchmark;
- Furthermore, to better fulfill task automation, it requires LLMs to derive multiple task elements (e.g., decomposed tasks, invoked tools, and tool parameters). These also compel us to devise more advanced evaluation metrics tailored to assess the performance of LLMs in these dimensions.

Therefore, in light of these observations, we found that the majority of existing benchmarks fall short of adequately showcasing the full potential of LLMs in autonomous task completion. For example, conventional benchmarks like GLUE (Wang et al., 2019b) or SuperGLUE (Wang et al., 2019a), encompass a lot of specific tasks to evaluate the capability of LLMs in a single scenario, while cannot well reflect the versatility of task automation. Some other researchers attempted to advocate for more rigorous benchmarks (e.g., MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), AGIEval (Zhong et al., 2023)) by involving more general scenarios (e.g., exams). But all of them can only reflect the capability of language skills, and are not able to manifest the capability of LLMs in task automation. Besides, how to conduct the evaluation for task automation is still a troublesome problem. Therefore, in this paper, we expect to develop a benchmark with appropriate evaluation metrics to better evaluate LLMs in task automation.

To this end, we present TASKBENCH to benchmark the capability of LLMs in the realm of task automation. Specifically, as aforementioned, the data collection for task automation requires us to consider different sophisticated settings, which makes it more challenging. However, just as shown in Figure 1, compared with directly simulating user requests, LLMs usually need to parse tasks for automation, and these parsed tasks (i.e., decomposed tasks, invoked tools, and parameters) are easier to collect and construct. Therefore, we raise a simple idea: *is it possible to synthesize user instruction based on the expected parsed tasks?*

To fulfill this, we first present the concept of Tool Graph (TG), which gathers diverse tools to address specific tasks. Specifically, every two tools in TG can have a connection if they have a dependency. The structure of the whole tool connections can considered as a graph. Therefore, to simulate user instruction, we can randomly sample a sub-graph from TG to represent the expected task list in user instruction and then apply a back-instruct strategy to generate the final user instruction. During the generation, we provide three different architectures for sampling to enable better controllability, which are node, chain, and directed acyclic graph (DAG). Moreover, a self-critic mechanism is appended to further refine the data quality of our dataset by reviewing their consistency. To guarantee diversity, our data generation is applied to three domains (e.g., Hugging Face (Wolf et al., 2019), multimedia, and daily life) to formulate our TASKBENCH for evaluating LLM in task automation.

After building the dataset, another challenge is how to effectively and quantitatively evaluate the capability of LLMs in task automation. We note that the primary steps of LLMs in automating tasks include task decomposition, tool selection, and parameter prediction. Therefore, we further propose an evaluation system, called TASKEVAL, which encompasses a series of metrics to provide objective evaluations to measure the capability of LLMs in task decomposition, tool selection, and predicting the parameters of tools. Moreover, we also conduct human evaluation to prove the positive correlation of our evaluation with human assessment.

Overall, the contributions of our paper can be summarized as:

- We introduce TASKBENCH, a new benchmark to support the development of LLM in task automation, which comprises a novel data generation to address the data deficiency in this area;
- We further present TASKEVAL to effectively and quantitatively evaluate the capability of LLMs in automating tasks from different aspects, including task decomposition, tool selection, and parameter predictions;
- The experimental results on different LLMs and additional dataset analysis demonstrate that our proposed TASKBENCH can effectively reflect the capability of LLMs in multiple dimensions with the support of TASKEVAL and show high correlations with human evaluation.

2 TASKBENCH DATASET

In this section, we introduce the construction of TASKBENCH, the benchmark meticulously designed to facilitate the development of LLMs in task automation. Specifically, unlike previous methods which use collection or instruction methods, TASKBENCH can consider the complex relationships among multiple tasks to simulate more practical and complex user instruction. Figure 2 illustrates the entire process of our method to build the datasets. More details will be introduced in the following subsections.

2.1 PRELIMINARY

Task automation aims to fulfill complex user instructions in real-world scenarios. In this setting, the user instructions could encompass multiple sub-tasks, and the execution of each sub-task can be completed by invoking a tool (Schick et al., 2023). Besides, there could also remain some temporal or resource dependencies among these sub-tasks. Therefore, we think that each user instruction can be represented as a combination of tools with connections like graph structure, just as shown in Figure 1. Consequently, we introduce the concept of the Tool Graph (TG), which will be used in our benchmark construction. The tool graph can be viewed as a structured representation that centers on tools with their dependency. Here, we assume a tool as t and denote a TG as $\mathcal{G} = \{T, D\}$, where $T = \{t_1, t_2, \ldots, t_n\}$ represents the collection of tools, and D is a collection of $\{(t_a, t_b)\}$ that means tool t_a exhibits a dependency on tool t_b. To some extent, the tool graph offers a novel approach to organizing tools, capturing the relationships between different tools more effectively than traditional taxonomy trees. In the next subsection, we will introduce how to build a tool graph and utilize it to formulate our benchmark.

2.2 DATASET CONSTRUCTION

To accomplish user intent, LLMs usually adopt a stepwise process (e.g., task decomposition→tool selection→parameter prediction) to analyze the user request and convert it into multiple executable tasks. Therefore, it is essential to construct the dataset and allow LLMs to evaluate their automation capability in the above process.

To guarantee that the generated user instructions could cover the expected tasks and dependencies, we adopt a back-instruct strategy to simulate data. More specifically, it can summarized as three steps: 1) we first collect a tool repository and build a tool graph $\mathcal{G}$ with a collection of tools and their dependencies; 2) then we sample a sub-graph from $\mathcal{G}$, to obtain a specified structure; 3) based on the sampled tool sub-graph, we use LLMs to generate user instruction via back-instruct. More details are introduced as below.

2.2.1 TOOL GRAPH CONSTRUCTION

Building a tool graph requires us to collect many standalone tools from different sources. When combining different tools together, the dependencies among tools could be diverse, encompassing resource dependencies, temporal dependencies, environment dependencies, and so on. In our research, we mainly investigate two of them: resource and temporal dependencies. For the former one, it means the two tools can have a connection if the input type of tool t_a can match the output type of tool t_b. For the latter one, we devise tool graphs that highlight temporal dependencies, allowing any two tools to be linked to illustrate their order. In this work, we choose three scenarios to build the datasets for our benchmark:

Hugging Face Hugging Face Wolf et al. (2019) provides a wide variety of AI models to cover massive tasks across language, vision, audio, video, and so on. Each task defined by Hugging Face can be viewed as a tool to address a specific task. Specifically, each tool in Hugging Face has determined the type of its input and output. Hence, if tool t_a and t_b have a connection, the input type of t_a should match the output type of t_b. Guided by this principle, we constructed Hugging Face's tool graph, comprising 23 tools and 225 edges.

Multimedia In contrast to the Hugging Face tools, which are tailored for AI tasks, the multimedia tools is broader in scope. It provides more user-centric tools like file downloader, video editor, and so on. The policy for tool connections is the same as the Hugging Face domain. Finally, we could construct a tool graph over multimedia tools with 40 nodes and 449 edges.

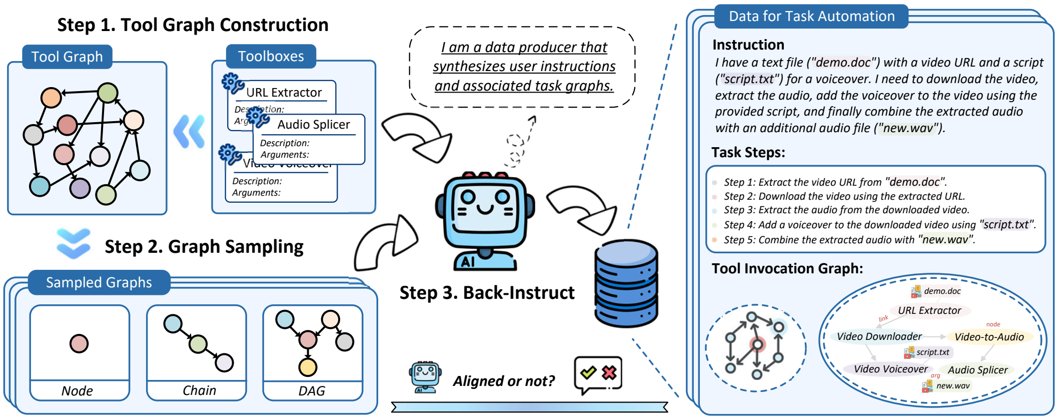
Daily Life APIs Sometimes, we also need some daily life services, including web search, shopping, and etc. Hence, these daily life APIs can also be considered as tools for specific tasks. However, it is worth noting that the type of dependencies among these APIs is predominantly temporal. Therefore, two daily life APIs have a successive order if they are connected. In this scenario, we can build a tool graph with 40 nodes and 1,560 edges.

Please note that our method's applicability extends beyond the scenarios mentioned above. We present more details about these tool graphs on different domains in Appendix A.10.

2.2.2 SAMPLING ON TOOL GRAPH

Based on the above steps, we can sample a sub-graph from the constructed TG and keep the connections of sampled tools from the TG to capture the dependencies between tools. Following the setting of HuggingGPT, we categorize the sub-structure of a TG into three types: node, chain, and directed acyclic graph (DAG). Each type embodies a specific pattern for tool invocation:

- **Node** represents standalone tool invocations, suitable for addressing simple tasks necessitating only a single tool.
- **Chain** corresponds to sequential tool invocations, where tools need to be stepwise executed to complete a task.
- **DAG** depicts more intricate tool invocations. A tool might rely on multiple preceding tools or influence several subsequent tools.



I am a data producer that synthesizes user instructions and associated task graphs.

Instruction

I have a text file ("demo.doc") with a video URL and a script ("script.txt") for a voiceover. I need to download the video, extract the audio, add the voiceover to the video using the provided script, and finally combine the extracted audio with an additional audio file ("new.wav").

Task Steps:

- *Step 1: Extract the video URL from "demo.doc".*
- *Step 2: Download the video using the extracted URL.*
- *Step 3: Extract the audio from the downloaded video.*
- *Step 4: Add a voiceover to the downloaded video using "script.txt".*
- *Step 5: Combine the extracted audio with "new.wav".*

Figure 2: Construction of the TASKBENCH: Initially, we transform the toolbox into a tool graph by creating connections between tools based on their dependencies. Subsequently, we sample diverse subgraphs from the tool graph, which may be single nodes, chains, or directed acyclic graphs. Utilizing the sampled tool subgraphs, we “back-instruct” the LLM to inversely craft user instructions, task steps, and tool invocation graphs. Furthermore, we implement critics to evaluate the consistency of the generated tool invocation graphs with the sampled tool subgraphs.

By sampling sub-graphs from these three substructures, we can emulate a variety of valid tool invocation patterns for user instruction. We represent the tool subgraph in $\mathcal{G}$ as $\mathcal{G}_s = \{T_s, D_s\}$, where $T_s = \{t_{s1}, t_{s2}, \ldots, t_{sk}\}$ with $k < n$ and $D_s = \{(t_{sa}, t_{sb})\}$, such that t_{sa} and t_{sb} belong to T_s. The sampling of the tool graph can be described as:

$$\text{Sample}(\mathcal{G}, \text{mode}, \text{size}) \rightarrow \mathcal{G}_s, \tag{1}$$

where the mode specifies the sampling mode (e.g., Nodes, Chains, DAGs), and the size indicates the number of tools (Here we set its range as $\{1, 2, ..., 10\}$). These factors determine the topological nature and magnitude of the tool sub-graph in user instructions, respectively.

2.2.3 BACK-INSTRUCT

Next, based on the sampled sub-graph $\mathcal{G}_s$, we use LLMs to synthesize user instructions. We term this process BACK-INSTRUCT, which can considered as a data engine to convert the sampled tools into user instruction. Specifically, given a sampled subgraph $\mathcal{G}_s$, we formulate the following BACK-INSTRUCT procedure, empowering LLMs to produce the corresponding instructions I:

$$\text{BackInstruct}_1(\mathcal{G}_s = (T_s, D_s)) \rightarrow I. \tag{2}$$

Here, the sampled sub-graph $\mathcal{G}_s$ can instruct LLMs to generate user requests covering these related sub-tasks, and further with their dependencies. Such a strategy ensures the complexity and quality of the generated data.

Specifically, we note that sampled sub-graphs can only provide information on tool invocation skeletons, lacking the critical parameters for tool execution. Therefore, based on the generated instruction I in Eqn. 2, we let the LLM to populate the parameters for the tool subgraphs and generate the final tool invocation graph $\bar{\mathcal{G}}$ along with the corresponding task decomposition steps P:

$$\text{BackInstruct}_2(\mathcal{G}_s = (T_s, D_s), I) \rightarrow \{P, \bar{\mathcal{G}}\}. \tag{3}$$

After that, we introduce a self-critic mechanism to check and filter out the generated instruction to guarantee quality. Here, we offer two variants: LLM-based and rule-based. The former aims to use LLM to check the alignments between the generated data and the sampled tool sub-graph. While the latter uses straightforward rules to determine the alignment between the tool graphs in created data and the sampled tool graphs. Here, we use the nodes and edges of the sampled graph to determine the consistency. Figure 2 illustrates each step of our data engine to simulate user instructions.

Based on the above steps, we build TASKBENCH across three domains, which use GPT-4 as the data engine. The ratio of different modes (i.e., Node, Chain, DAG) is set as $3:7:8$ for sampling and the ratio for the number of different tools is set as $\{0.1, 0.2, 0.3, 0.2, 0.1, 0.05, 0.025, 0.025, 0.025\}$. More detailed designs about our data engine and the statistics of the constructed datasets are provided in the Appendix A.8.

2.3 EVALUATION OF THE DATASET QUALITY

To demonstrate the quality of TASKBENCH datasets, we conducted in-depth human evaluations based on generated samples. We also conduct a case study and error analysis on the constructed datasets. Please refer to Appendix A.3 and Appendix A.4 for more details.

Evaluation Metrics To assess the quality of datasets constructed by Back-Instruct, we designed three metrics in our evaluation criteria. Two measure the quality of instructions, and one evaluates tool invocation graphs:

- Metrics for Instruction:
  - **Naturalness:** This metric measures the reasonableness of the instructions, including the commonality of dependencies between tools and their alignment with real-world needs.
  - **Complexity:** This metric assesses the complexity of the instructions, considering factors such as task depth, the number of involved tools, and the relationships between these tools.
- Metric for Tool Invocation Graphs:
  - **Alignment:** This measures how well the tool invocation graphs align with the instructions, i.e., whether the tool invocation graphs can effectively address the user's commands.

Each metric is scored from 1 to 5, and we design these metrics to assess the effectiveness and faithfulness of our TASKBENCH in task automation.

Comparison with Baselines To make a fair comparison, we choose two additional baselines to compare our Back-Instruct:

- **Back-Instruct (Ours):** we sample tool subgraphs and then backtranslate to instructions and further refine the tool invocation graph.
- **Back-Instruct w/o edges:** compared with our Back-Instruct, we eliminated edges from our sampled tool subgraphs, preserving only the tool node information in the prompt.
- **Self-Instruct:** (Wang et al., 2023b) based on manually labeled demonstrations and all tools with descriptions, we directly employed GPT-4 to autonomously select tools and then generate the instructions with tool invocation graphs.

Table 1: Human evaluation (rating from 1 to 5) on samples constructed by different methods. Average score rating from three human experts.

Methods	Naturalness↑	Complexity↑	Alignment↑	Overall↑
Back-Instruct	**3.89**	**4.01**	**3.66**	**3.85**
Back-Instruct w/o edges	3.44	3.27	3.62	3.44
Self-Instruct	2.18	2.01	3.64	2.61

Evaluation Results During the human evaluation, we randomly selected 50 samples from our TASKBENCH and invited three domain experts to assess the quality of these samples. To ensure a fair and unbiased evaluation, all samples will be anonymized. We provide canonical samples for these experts to calibrate their criteria during the annotations, and calculate an average score of all experts' ratings as the final results. All results can be found in Table 1.

We observed that all methods (Self-Instruct or Back-Instruct) can guarantee the alignment. However, our method, Back-Instruct, scored highest in Naturalness and Complexity. We attribute these superiorities to the realistic resource or temporal dependencies in the sampled tool subgraphs, which allow us to generate more natural instructions in complex scenarios (e.g., multi-tools utilization).

3 TASKEVAL

We could collect ample samples (i.e., synthetic user instructions) with annotations (i.e., sampled tool sub-graph) to evaluate the capability of LLMs in automating tasks. Here, we introduce TASKEVAL,

Table 2: Evaluation for tool selection. Node F1 *(n-F1)* and Edge F1 *(e-F1)* for node and edge prediction. For nodes, a prediction is deemed positive if the predicted node's ID aligns with any of the ground-truth node labels. For edges, both the source and target nodes of a predicted edge must correspond exactly. Normalized Edit Distance *(NED)* measures the normalized number of operations required to correct the prediction for chain structure.

TOOL SELECTION - Predicts tools and their dependencies.

	LLM	**Node**	**Chain**			**DAG**		**Overall**	
		n-F1 ↑	*n-F1*↑	*e-F1* ↑	*NED* ↓	*n-F1*↑	*e-F1* ↑	*n-F1*↑	*e-F1* ↑
Hugging Face Tools	gpt-4	84.34	80.79	55.73	39.70	82.86	56.39	81.54	54.70
	gemini-pro	77.46	76.12	45.51	43.10	79.05	49.36	76.62	43.50
	claude-2	69.83	80.67	48.11	40.03	84.52	53.40	79.00	43.51
	gpt-3.5-turbo	56.91	72.63	39.92	46.52	73.79	38.55	69.49	33.36
	text-davinci-003	40.71	66.05	36.04	48.57	64.64	34.19	59.38	29.37
	codellama-13b	43.68	55.65	17.80	62.23	52.87	13.19	53.16	14.64
	nous-hermes-13b	58.66	52.39	9.01	62.48	51.99	6.33	53.62	8.29
	vicuna-13b-v1.5	51.74	50.37	8.40	66.83	52.46	9.06	50.82	7.28
	llama-2-13b-chat	43.59	49.87	8.22	64.99	49.60	9.11	48.47	7.30
	wizardlm-13b	54.69	54.50	2.22	60.55	52.93	0.92	54.40	2.05
Multimedia Tools	gpt-4	97.13	89.70	69.29	28.93	92.32	71.64	90.90	69.27
	gemini-pro	73.61	82.65	55.50	35.62	85.29	57.80	81.54	52.07
	claude-2	66.16	83.95	59.22	33.41	82.98	54.28	80.94	53.01
	text-davinci-003	59.15	76.87	50.79	38.54	79.00	50.69	73.97	45.81
	gpt-3.5-turbo	53.55	76.81	50.30	39.05	78.65	49.52	72.83	44.02
	codellama-13b	43.70	66.89	28.77	46.35	68.68	28.79	62.78	24.61
	vicuna-13b-v1.5	66.64	59.18	16.49	54.17	61.40	13.95	60.61	14.78
	nous-hermes-13b	60.58	58.53	9.47	56.02	59.39	9.57	58.97	8.90
	wizardlm-13b	55.13	50.57	4.92	58.46	49.38	5.52	51.24	4.82
	llama-2-13b-chat	38.02	45.14	1.62	65.29	45.95	2.11	43.87	1.63
Daily Life APIs	gpt-4	95.97	97.06	83.47	38.69	96.41	42.01	96.91	80.53
	claude-2	79.57	95.36	80.68	39.93	93.85	41.04	93.52	75.31
	gemini-pro	76.15	92.79	64.58	41.64	89.68	28.42	90.75	59.45
	gpt-3.5-turbo	52.18	90.80	70.66	43.50	86.94	30.85	85.37	60.67
	text-davinci-003	68.49	82.15	60.12	47.14	76.81	24.54	80.42	54.90
	codellama-13b	89.75	87.80	65.92	44.42	83.61	27.47	87.73	63.16
	wizardlm-13b	92.27	65.74	14.51	55.80	63.80	9.20	69.34	14.18
	vicuna-13b-v1.5	90.59	73.74	13.24	51.43	67.92	5.62	75.67	12.48
	nous-hermes-13b	92.50	71.17	3.55	53.47	70.65	2.86	73.45	3.50
	llama-2-13b-chat	34.11	57.61	20.13	67.06	56.18	8.42	55.77	17.02

which encompasses a series of evaluation metrics to measure LLMs in multiple dimensions, including task decomposition, tool selection, and parameter prediction. To simulate the process of LLMs in automating tasks, we adopt a standard prompt for each LLM, which enables it to first disassemble user requests into multiple sub-tasks (i.e., task decomposition), and then predict tool invocations with their parameters and task dependencies to generate a tool invocation graph. Based on the built datasets and the standard inference process, we design pertinent metrics to evaluate three stages (i.e., task decomposition, tool selection, and parameter predictions) in task automation. Here, we choose the GPT family (Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023) and open-source LLMs (Touvron et al., 2023a; Chiang et al., 2023; Rozière et al., 2023; Xu et al., 2023; Yang et al., 2023) as our main evaluation. Please see the Appendix A.7 for full evaluations of other open-source LLMs (Team, 2023a; Li et al., 2023a; Team, 2023b).

3.1 TASK DECOMPOSITION

Task decomposition is a pivotal component of task automation. By decomposing user instruction into a sequence of executable sub-tasks, the autonomous agent can more effectively fulfill user intent. During the task decomposition, each step will generate textual descriptions. Here, we use three subjective metrics to measure the quality in analyzing sub-tasks: **Rouge-1** ***(R1)***, **Rouge-2** ***(R2)***,

Table 3: Evaluation for parameter prediction of tools. *t-F1* evaluate the pair of (task, parameter name), *v-F1* evaluate the triple of (task, parameter name, parameter value).

TOOL PARAMETER PREDICTION - Predicts parameters for the tool execution.

	LLM	Node		Chain		DAG		Overall	
		t-F1↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑
Hugging Face Tools	gpt-4	80.05	74.10	76.66	58.15	78.24	60.03	77.31	60.86
	gemini-pro	67.63	56.54	66.60	46.35	70.41	50.56	67.12	48.54
	claude-2	48.07	32.14	66.35	45.57	68.59	48.19	63.00	43.08
	text-davinci-003	38.51	27.43	56.90	38.76	57.03	38.90	52.53	36.04
	gpt-3.5-turbo	37.70	19.81	60.96	41.15	61.33	42.89	55.88	36.32
	codellama-13b	20.09	12.58	36.40	21.31	33.43	20.48	32.06	18.87
	nous-hermes-13b	46.38	31.06	35.55	13.81	33.06	13.69	37.51	17.66
	wizardlm-13b	43.97	25.90	37.34	12.48	38.43	13.79	38.76	15.35
	llama-2-13b-chat	29.80	20.54	32.14	13.57	32.16	15.23	31.61	15.38
	vicuna-13b-v1.5	25.71	13.11	28.99	11.14	30.04	13.60	28.34	11.85
Multimedia Tools	gpt-4	95.64	87.12	85.60	69.83	87.57	72.79	87.06	72.31
	gemini-pro	62.21	50.48	72.99	55.21	76.13	58.79	71.67	54.82
	claude-2	53.81	24.02	75.60	58.12	72.41	52.43	71.63	51.58
	gpt-3.5-turbo	44.94	11.96	70.53	47.76	71.82	47.95	65.91	40.80
	text-davinci-003	60.30	20.78	69.91	44.76	71.91	45.76	68.48	40.70
	codellama-13b	32.01	16.10	52.30	32.51	53.08	33.79	48.19	29.13
	vicuna-13b-v1.5	52.72	35.55	39.31	21.00	40.05	21.40	41.62	23.62
	nous-hermes-13b	50.11	37.80	41.98	17.89	43.99	20.04	43.60	21.69
	wizardlm-13b	49.79	33.59	36.88	14.87	36.61	18.68	39.10	18.74
	llama-2-13b-chat	28.49	17.01	30.26	9.66	31.00	11.35	29.99	11.32
Daily Life APIs	gpt-4	95.83	76.21	97.23	70.67	95.95	69.65	97.02	71.14
	claude-2	78.12	59.43	94.72	65.30	91.83	66.39	92.71	64.72
	gemini-pro	69.88	45.41	91.66	57.93	88.50	53.91	88.95	56.22
	gpt-3.5-turbo	43.81	28.77	89.21	61.11	83.88	56.13	81.97	55.66
	text-davinci-003	61.68	45.53	80.68	54.54	76.51	51.91	78.37	53.40
	codellama-13b	86.34	71.20	84.31	61.51	80.42	60.18	84.26	62.38
	vicuna-13b-v1.5	83.63	67.71	61.80	44.54	57.14	41.72	64.27	47.31
	nous-hermes-13b	79.69	63.29	62.64	45.32	63.26	45.74	64.47	47.22
	wizardlm-13b	89.27	72.96	50.68	36.48	49.03	35.75	55.00	40.53
	llama-2-13b-chat	10.39	7.32	38.89	25.37	36.43	23.40	35.11	22.94

and **BertScore F1** ***(BsF)*** (Zhang et al., 2019). The results are reported in Table 11. We observe that the GPT-4 model significantly outperforms the open-source LLM model in task decomposition, achieving approximately 20%+ higher than others in Rouge-1/2. Moreover, we find that codellama-13b achieves the closest performance to the GPT family. We attribute the substantial code pre-training endowing it with a higher-level task decomposition capability compared to other LLMs.

3.2 TOOL SELECTION

The graph of tool invocation can be viewed as a concrete representation of task steps in user instruction, specifying the appropriate tool for each step. To orchestrate external tools effectively, the tool invocation graph should provide these pieces of information: 1) the dependency between tools to guarantee the order of executable sub-tasks; 2) the parameters that tools require. Therefore, it is necessary for us to measure the capability of LLMs in these aspects. Here, we first evaluate the predicted graph structure to measure the capability of LLMs in this subsection.

In a tool invocation graph, we denote the nodes as tools and the edge as the dependency between two tools. Therefore, we can evaluate the nodes and edges to measure the capability of LLMs in tool selection. Here, we crafted two distinct metrics: **Node F1** ***(n-F1)*** for node prediction and **Edge F1** ***(e-F1)*** for edge prediction. We also introduced the **Normalized Edit Distance** ***(NED)*** metric for chain structure, quantifying the adjustments needed to correct predictions. The results are detailed in Table 2 revealing that predicting edges in the tool invocation graph is more challenging than predicting nodes, with an F1 score difference of about 30% across all LLMs. Furthermore, we also observe that tasks with varying structures pose different challenges for LLMs. For instance, on simple node structures, open-source LLMs match the performance of gpt-3.5-turbo and text-davinci-003, but lag behind on more complex tasks. Overall, these designed metrics can effectively help us to better measure the capability of LLMs in task automation.

3.3 PARAMETER PREDICTION

We further divide the evaluation of the tool parameter prediction as twofold. The first metric is **Parameter Name F1** ***(t-F1)*** to evaluate the capability of LLMs in predicting the parameter of the tools. Another one is the **Parameter Name & Value F1** ***(v-F1)*** to measure both the parameter and its value. The results can be found in Table 3. We found that the precise prediction of parameters can determine the successful execution of tools, and the precise prediction of both parameters and values can guarantee the correctness of the tool invocation. Besides, we found that GPT-4 can significantly outperform open-source LLMs, which achieve *v-F1* scores of 60.86%, 72.31%, and 71.14% across the three domains. These results also highlight the limitations of current open-source LLMs in task automation, and which parts should be enhanced.

3.4 ANALYSIS

Factors Contributing to Task Automation Performance Our analysis identifies two primary factors impacting the performance of LLMs in task automation: 1) ***Reasoning Capabilities***: The ability to solve complex problems and reason effectively varies across LLMs, significantly impacting task automation. For instance, the GPT series exhibits superior reasoning in mathematical and coding tasks, reflecting advanced task planning and tool utilization skills. 2) ***Instruction Following***: Models fine-tuned for instruction following, like Vicuna-13b and WizardLLM-13b, outperform the baseline Llama-2-13b. Notably, WizardLLM-13b surpasses Vicuna-13b due to its more sophisticated instruction fine-tuning, highlighting the importance of precise instruction adherence.

Intrinsic Differences in LLMs in Performing Task Automation Our findings suggest that: 1) ***Code Pre-training***: Models with extensive code pre-training, such as Code-Llama, surpass other LLMs in task automation. Our data shows an average improvement of 4.45% in tool prediction and 12.76% in parameter prediction across various domains, underscoring the necessity of structured text for connecting automation stages. 2) ***Alignment***: Models employing human alignment techniques (e.g., the GPT series with RLHF) show enhanced task automation capabilities compared to their open-source counterparts, indicating that RLHF promotes more generalized reasoning skills and mitigates instruction-specific overfitting.

3.5 CONSISTENCY WITH HUMAN EVALUATION

To ensure the reliability of TASKBENCH evaluations, we further investigate their consistency (measured by Kendall's τ and Spearman's ρ) with human evaluations. From Table 8, we find that the average values for Kendall's τ and Spearman's ρ are 0.89 and 0.78, respectively. This indicates a very positive correlation between human evaluation and our TASKBENCH, which further validates the effectiveness of our proposed benchmark for task automation.

4 CONCLUSION

In this paper, we introduce TASKBENCH, a benchmark to evaluate LLMs for task automation. More in detail, we first summarize three critical stages for LLMs in automating tasks, which include task decomposition, tool selection, and parameter prediction for tools. The performance of these three stages reflects the task automation capabilities of LLMs, and thus we expect to construct evaluation datasets for them. To fulfill this, we introduce the concept of ToolGraph, which collects different tools with their connections, and then adopt a back-instruct method to simulate user instruction based on the sampled sub-graph from ToolGraph. Based on our curated datasets, we further introduce TASKEVAL to systematically evaluate the capability of LLMs in automating tasks, including task decomposition, tool selection, and parameter prediction. Experimental results demonstrate our

TASKBENCH can be effectively utilized to evaluation LLMs in task automation. In the future, we will extend our benchmark to various domains and design more advanced metrics to further explore the potential of LLMs in task automation and build powerful autonomous agents.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (No. 62376245), the Key Research and Development Program of Zhejiang Province, China (No. 2024C01034), the project of the Donghai Laboratory (Grant no. DH-2022ZY0013), and MOE Engineering Research Center of Digital Library.

REFERENCES

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. Palm 2 technical report. *CoRR*, abs/2305.10403, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, pp. 1–25, 2020.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

Significant Gravitas. Auto-gpt: An autonomous gpt-4 experiment. `https://github.com/Significant-Gravitas/Auto-GPT`, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. MetaGPT: Meta Programming for Multi-Agent Collaborative Framework, August 2023.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, , and Hao Zhang. How long can open-source llms truly promise on context length?, June 2023a. URL `https://lmsys.org/blog/2023-06-29-longchat`.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for ”Mind” Exploration of Large Scale Language Model Society, March 2023b.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 2023c.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, Yun Wang, Linjun Shou, Ming Gong, and Nan Duan. Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis. *CoRR*, abs/2303.16434, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *CoRR*, abs/2308.03688, 2023.

Yohei Nakajima. Babyagi. `https://github.com/yoheinakajima/babyagi`, 2023.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, pp. 27730–27744, 2022.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior, April 2023.

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

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

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

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving AI tasks with chatgpt and its friends in huggingface. *CoRR*, abs/2303.17580, 2023.

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

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

InternLM Team. Internlm: A multilingual language model with progressively enhanced capabilities. `https://github.com/InternLM/InternLM`, 2023a.

MosaicML NLP Team. Introducing mpt-7b: A new standard for open-source, ly usable llms, 2023b. URL `www.mosaicml.com/blog/mpt-7b`. Accessed: 2023-03-28.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023b.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, pp. 3261–3275, 2019a.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019b.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, May 2023a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL `https://aclanthology.org/2023.acl-long.754`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

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

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

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. *CoRR*, abs/2304.06364, 2023.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for LLM question answering with external tools. *CoRR*, abs/2306.13304, 2023.

A APPENDIX

A.1 RELATED WORKS

Large language models (ChatGPT (Ouyang et al., 2022), GPT-4 (OpenAI, 2023), LLAMA (Touvron et al., 2023a;b), Bard (Anil et al., 2023)) have drawn the development of autonomous agents (e.g., AutoGPT (Gravitas, 2023), HuggingGPT (Shen et al., 2023), BabyAGI (Nakajima, 2023)). These applications can be considered as a form of task automation, which uses LLMs as the controller to analyze user instructions and search for the most suitable solution (e.g., external models) to obtain answers. Despite these advances in this area, it still lacks a systematic and standardized benchmark to measure the capability of LLMs in automation tasks. Traditional benchmarks like GLUE (Wang et al., 2019b) or SuperGLUE (Wang et al., 2019a) can only evaluate the capability of pre-trained models in a single task. To further explore the capability of LLMs, some benchmarks (AlpacaEval (Li et al., 2023c), AGIEval (Zhong et al., 2023), GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2021) and etc) are introduced to explore the capability of LLMs in solving problems, but they can only manifest the language skills and are inadequate to test the capability in automating tasks. Recently, some works attempted to introduce some benchmarks to facilitate the development of this area. For example, APIBench is proposed by Gorilla (Patil et al., 2023) which uses self-instruct (Wang et al., 2023b) to generate data from API calls. ToolAlpaca (Tang et al., 2023), ToolQA (Zhuang et al., 2023) and ToolBench (Qin et al., 2023) respectively introduce different benchmarks from the perspective of tool utilization. Besides, AgentBench (Liu et al., 2023) is a benchmark to explore the capability of LLMs in different environments, which is another important part of autonomous agents. In this paper, we focus on multiple dimensions (e.g., task decomposition, tool selection, and parameter prediction) to build a multifaceted benchmark, including dataset and evaluation, to comprehensively evaluate the capability of LLMs in task automation.

A.2 DISCUSSION WITH RELATED WORKS

We also note that there are some concurrent works (e.g., ToolBench, APIBench, etc) to investigate this area. Therefore, we also conducted an in-depth discussion to analyze the differences between TASKBENCH and these works. Generally, we summarize these differences from the perspectives of datasets and evaluations:

Datasets Currently, all of these works will adopt a self-instruct strategy to generate instructions to simulate complex user instructions, which requires planning, tool utilization, and etc. However, most of them still lack the annotations of ground truth, and only rely on human evaluation to check the quality of LLM generations. Some works like ToolBench additionally use a depth-first search-based decision tree to search a solution path annotation. But such a strategy will also bring some new issues like costs and hallucination or bias in annotations. Compared with them, due to the design of the Tool Graph, TASKBENCH can guarantee the ground truth and demonstrate these advantages:

- **Efficiency**: Our method requires only one API call to generate a complete sample (creating instructions, generating a tool invocation graph, and checking). In contrast, ToolLLM requires one API call to generate instructions and an average of four inference steps during DFS for annotation. Additionally, ToolLLM uses few-shot learning, which consumes more tokens than our zero-shot approach.
- **Higher Quality of Instructions**: Our tool graph includes potential tool invocation relationships (resource and temporal dependencies). Based on the tool graph, the instructions generated by ChatGPT are more in line with actual human behavior. As we demonstrated in Appendix A.3, the generation based on a tool graph with edges significantly improves Naturalness, Complexity, and Alignment compared to generation without edges.
- **Stability**: ToolLLM's instruction generation might not cover all sampled instructions based on the API set. Our method not only covers all sampled tools but also strictly follows the dependency between tools. We can control instruction generation through the sampling process, such as the number of tools and the topology of the tool graph.
- **Reliable Annotations**: In ToolLLM, instruction generation, and tool invocation annotation are independent processes. However, in our approach, the sampled tool graph can directly be utilized as the final annotations for the assessment. Hence, our Back-Instruct can directly generate

instructions with better alignments to the annotations and does not need to generate answers anymore. Besides, we also utilize the consistency between the sampled tool graph and the generated tool invocation graph to further filter out low-quality annotations, ensuring high-quality tool graph annotations.

Evaluation During the evaluation, most of them usually use human evaluation to measure the capability of LLMs, which requires massive human labor. Some works follow the AlpacaEval setting to measure the capability of LLMs in tool utilization. Compared with these settings, Our evaluation is composed of both subjective and objective evaluations, which also encompasses multiple aspects, including task decomposition, tool prediction, and parameter prediction. For each aspect, we have developed multiple well-designed metrics to assess the different capabilities required for task automation. Besides, we also conduct human evaluations to investigate the performance of different LLMs and demonstrate the correlations with the subjective evaluation metrics.

A.3 CASE STUDY OF BACK-INSTRUCT

We draw some cases to intuitively show the differences between Back-Instruct, Back-Instruct w/o edges and Self-Instruct (Wang et al., 2023b), as shown in Table 4. From these examples, we observe that our Back-Instruct can generate examples with more comprehensive and interconnected tool usage, reflecting higher naturalness and complexity in instruction generation.

A.4 ERROR ANALYSIS

A.4.1 ERROR ANALYSIS ON TASKBENCH DATASET

Despite the advanced instruction generation and labeling capabilities of gpt-4, we admit that it is challenging to guarantee the correctness of all generated samples. To better understand our dataset and assess its accuracy, we conduct human evaluations to provide a thorough error analysis. Here, we first randomly sampled 148 samples, and our labeling team identified 18 error samples (nearly 12%) from the sampled data. We attribute these incorrect samples to five distinct error categories. Typical examples and the proportions for each category are shown in Table 5 and Figure 3:

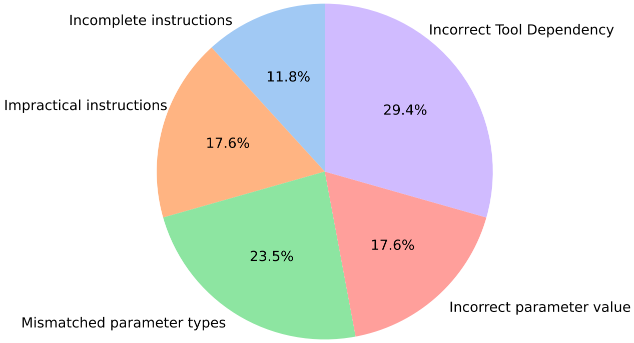


Figure 3: Error Analysis on TASKBENCH

- **Incorrect Instructions**:
  - **Incomplete instructions**: This error occurs when the instructions lack the necessary details or resources for successful completion.
  - **Impractical instructions**: The instructions could be irrational or impossible to execute with the capabilities of current tools.
- **Parameter Errors:**
  - **Mismatched parameter types**: This error occurs when the parameters provided do not match the expected types for the used tool.
  - **Incorrect parameter value**: This error is evident when the values provided for the parameters are incorrect or not suitable for the task.
- **Incorrect Tool Dependency**: This error type refers to the incorrect linking or sequencing of tools required for a task.

Based on these observed errors, we conclude that it is necessary to build a more elaborate prompt (e.g., more detailed tool-use specification and demonstrations) to describe tool parameters and tool

Table 4: Comparative analysis of Back-Instruct, Back-Instruct w/o edges, and Self-Instruct.

Method	Tools	Instruction	Tool Invocation Graph
Back-Instruct	”sampled_nodes”: [{”id”: ”Audio–to–Image”}, {”id”: ”Image Colorizer”}, {”id”: ”Image Stitcher”}, {”id”: ”Image Style Transfer”}, {”id”: ”Image–to–Text”}], ”sampled_links”: [{”source”: ”Image Style Transfer”, ”target”: ”Image Stitcher”}, {”source”: ”Audio–to–Image”, ”target”: ”Image Colorizer”}, {”source”: ”Image Colorizer”, ”target”: ”Image Style Transfer”}]	I have an image (example.jpg) with some text on it and an audio file (example.wav). I’d like you to extract the text from the image, generate a visual representation of the audio file, colorize this visual representation using the colors from the original image, apply the visual style of the colorized image to the visual representation of the audio, and finally, stitch these two images together to create a panorama or collage.	{”nodes”: [{”id”: ”Audio–to–Image”, ”input”: [”example.wav”]}, {”id”: ”Image Colorizer”, ”input”: [”Audio–to–Image”]}, {”id”: ”Image Stitcher”, ”input”: [”example.jpg”, ”Image Style Transfer”]}, {”id”: ”Image Style Transfer”, ”input”: [”Image Colorizer”, ”Audio–to–Image”]}, {”id”: ”Image–to–Text”, ”input”: [”example.jpg”]}], ”links”: [{”source”: ”Audio–to–Image”, ”target”: ”Image Colorizer”}, {”source”: ”Image Colorizer”, ”target”: ”Image Style Transfer”}, {”source”: ”Image Style Transfer”, ”target”: ”Image Stitcher”}]}
Back-Instruct w/o edges	”sampled_nodes”: [{”id”: ”Audio–to–Image”}, {”id”: ”Image Colorizer”}, {”id”: ”Image Stitcher”}, {”id”: ”Image Style Transfer”}, {”id”: ”Image–to–Text”}]	I would like to create a stylized collage from ’example.jpg’ and the sound of the ’example.wav’ file. The visual style should be inspired by the similar image search result. The collage should be colorized, and I would like any text in the final image to be extracted.	{”nodes”: [{”id”: ”Audio–to–Image”, ”input”: [”example.wav”]}, {”id”: ”Image Colorizer”, ”input”: [”Image Style Transfer”]}, {”id”: ”Image Search (by Image)”, ”input”: [”example.jpg”]}, {”id”: ”Image Stitcher”, ”input”: [”Audio–to–Image”, ”example.jpg”]}, {”id”: ”Image Style Transfer”, ”input”: [”Image Stitcher”, ”Image Search (by Image)”]}, {”id”: ”Image–to–Text”, ”input”: [”Image Colorizer”]}], ”links”: [{”source”: ”Audio–to–Image”, ”target”: ”Image Stitcher”}, {”source”: ”Image Colorizer”, ”target”: ”Image–to–Text”}, {”source”: ”Image Search (by Image)”, ”target”: ”Image Style Transfer”}, {”source”: ”Image Stitcher”, ”target”: ”Image Style Transfer”}, {”source”: ”Image Style Transfer”, ”target”: ”Image Colorizer”}]}
Self-Instruct	”sampled_nodes”: [{”id”: ”Audio–to–Image”}, {”id”: ”Image Colorizer”}, {”id”: ”Image Stitcher”}, {”id”: ”Image Style Transfer”}, {”id”: ”Image–to–Text”}]	I have a black and white image of an old newspaper (example.jpg) and I want to colorize it first, then apply the style of a modern newspaper (example.png) to it, and finally extract the text from the stylized image.	{”nodes”: [{”id”: ”Image Colorizer”, ”input”: [”example.jpg”]}, {”id”: ”Image Style Transfer”, ”input”: [”Image Colorizer”, ”example.png”]}, {”id”: ”Image–to–Text”, ”input”: [”Image Style Transfer”]}], ”links”: [{”source”: ”Image Colorizer”, ”target”: ”Image Style Transfer”}, {”source”: ”Image Style Transfer”, ”target”: ”Image–to–Text”}]}

dependencies when generating the tool invocation graph. Besides, we will also introduce more high-quality criteria to continuously improve our dataset in addition to our rule-based and LLM-based critics.

Table 5: Error Analysis on TaskBench.

Error Type	Example
Incomplete instructions	I have a long text and I would like to get a summarized version of it, then generate an image that represents the main idea of the summarized text.
Impractical instructions	I have a text: ’This training vid is amazing! Speed it up by 1.5x please!’. Analyze the sentiment, expand it, find the video URL and adjust the video speed.
Mismatched parameter types	I want to find articles related to climate change and analyze their sentiment. Please translate non-English articles to English. `{"task_nodes": [{"task": "Text Search", "arguments": ["climate change"]}, {"task": "Text Sentiment Analysis", "arguments": ["<node-0>"]}, {"task": "Text Translator", "arguments": ["<node-1>"]}], "task_links": [{"source": "Text Search", "target": "Text Sentiment Analysis"}, {"source": "Text Sentiment Analysis", "target": "Text Translator"}]}`
Incorrect parameter value	I have two audio files from online lectures at the following URLs: ’example1.wav’ and ’example2.wav’. I want them combined into a single audio file, transcribe the speech into text, and check the text for grammatical errors. `{"task_nodes": [{"task": "Audio Downloader", "arguments": ["example1.wav", "example2.wav"]}, {"task": "Audio Splicer", "arguments": ["<node-0>"]}, {"task": "Audio-to-Text", "arguments": ["<node-1>"]}, {"task": "Text Grammar Checker", "arguments": ["<node-2>"]}], "task_links": [{"source": "Audio Downloader", "target": "Audio Splicer"}, {"source": "Audio Splicer", "target": "Audio-to-Text"}, {"source": "Audio-to-Text", "target": "Text Grammar Checker"}]}`
Incorrect Tool Dependency	I want to create a more engaging version of this short text: ’Join us for a fun-filled evening!’ and find some videos related to its sentiment. `{"task_nodes": [{"task": "Article Spinner", "arguments": ["<node-2>"]}, {"task": "Text Expander", "arguments": ["Join us for a fun-filled evening!"]}, {"task": "Text Sentiment Analysis", "arguments": ["<node-1>"]}, {"task": "Video Search", "arguments": ["<node-2>"]}], "task_links": [{"source": "Text Expander", "target": "Text Sentiment Analysis"}, {"source": "Text Sentiment Analysis", "target": "Article Spinner"}, {"source": "Text Sentiment Analysis", "target": "Video Search"}]}`

Table 6: Error Distribution in Different LLMs.

	Required Tool Missing	Tool Dependency Error	Tool Parameter Error
gpt-4	0	2	3
gpt-3.5-turbo	2	8	11
code-llama-13b	4	9	13

A.4.2 Error Analysis of Different LLMs in Predicting Tool Invocation Graph

We analyze the failures in predicting the tool invocation graph that occur during task automation inference. These failures can be categorized into three main groups: incorrect tool names, incorrect tool dependencies, and mismatched tool parameters. For our analysis, we randomly selected 50 predictions, and the distribution of each error type across different LLMs is detailed in Table 6. We observed that:

- gpt-4 demonstrates the fewest errors in all categories, indicating a higher accuracy in predicting the tool invocation graph.
- gpt-3.5-turbo and code-llama-13b show a progressively higher number of errors. Notably, the ‘Incorrect Tool Dependency’ is the most common across all models, highlighting the challenge LLMs face in predicting accurate parameters for tools.

Further, we present specific cases in Table 7 to elucidate the nature of prediction errors in these LLMs. Given the following example, gpt-4 correctly interpreted the task in the given example, underscoring its advanced task automation capabilities. Conversely, gpt-3.5-turbo and code-llama-13b omitted a critical tool (‘Audio Downloader’), resulting in a ‘Missing Required Tool’ error. Additionally, code-llama-13b encountered compounded errors, including ‘Tool Parameter Error’ and ‘Incorrect Tool Dependency’.

```
Instruction:
I need an audio file downloaded from ’https://www.example.com/
example.wav’, then please reduce the background noise and apply a
reverb effect according to my instruction ’reverb 50\%’. Finally,
combine it with the audio file ’example.wav’.
Gold tool invocation graph:
"task_nodes": [
{"task": "Audio Downloader", "arguments": ["https://www.example.
com/example.wav"]},
{"task": "Audio Noise Reduction", "arguments": ["<node-0>"]},
{"task": "Audio Effects", "arguments": ["<node-1>", "reverb
50%"]},
{"task": "Audio Splicer", "arguments": ["<node-2>", "example.wav
"]}]
```

A.5 Metrics for Ranking Consistency

To compute the consistency of two rankings where the number of observations is n, we introduce two correlation coefficients: Spearman’s ρ and Kendall’s τ. In our work, they refer to the human and TaskBench rankings of large language models in terms of task automation capabilities.

Spearman’s ρ measures the strength and direction of the rank association between two variables. To calculate Spearman’s ρ, start by assigning ranks to each observation in both sets of data. For any tied values, assign the average rank. Next, compute the difference in ranks between the two datasets for each observation, and then square these differences. The coefficient is calculated as follow:

$$\rho = 1 - \frac{6 \times \text{ sum of squared rank differences}}{n\left(n^2-1\right)} \tag{4}$$

Kendall’s τ is calculated based on the consistency and inconsistency of pairs between two rankings. For both rankings, we will consider all possible pairs of items in them. For each pair of items, if the relative position is correct in both rankings, then we call this a “consistent pair”. If the relative position is wrong, then we call this an “inconsistent pair”.

$$\tau = \frac{(\text{number of concordant pairs }) - (\text{ number of discordant pairs })}{n(n-1)/2} \tag{5}$$

Table 7: Case Study of Prediction Errors in LLMs.

LLM	Predicted Tool Invocation Graph	Error Type
gpt-4	`{"task_nodes": [{"task": "Audio Downloader", "arguments": ["https://www.example.com/example.wav"]}, {"task": "Audio Noise Reduction", "arguments": ["<node-0>"]}, {"task": "Audio Effects", "arguments": ["<node-1>", "reverb 50%"]}, {"task": "Audio Splicer", "arguments": ["<node-2>", "example.wav"]}]}`	None
gpt-3.5-turbo	`{"task_nodes": [{"task": "Audio Noise Reduction", "arguments": [""https://www.example.com/example.wav""]}, {"task": "Audio Effects", "arguments": ["<node-0>", "reverb 50%"]}, {"task": "Audio Splicer", "arguments": ["<node-1>", "example.wav"]}]}`	Missing Required Tool
code-llama-13b	`{"task_nodes": [{"task": "Audio Noise Reduction", "arguments": ["example.wav"]}, {"task": "Audio Effects", "arguments": ["<node-0>", "reverb 50%"]}, {"task": "Audio Splicer", "arguments": ["<node-1>", "<node-0>"]}]}`	Missing Required Tool Tool Parameter Error Incorrect Tool Dependency

Table 8: Alignment of TASKBENCH with human evaluation. Kendall's τ alculates the proportion of aligned pairs, while Spearman's ρ measures the correlation between the ranks of elements.

Correlation Metric	Hugging Face Tools	Multimedia Tools	Daily Life APIs	Average
Kendall's τ	0.89	0.83	0.94	0.89
Spearman's ρ	0.78	0.62	0.93	0.78

Ranking Consistency Between TASKBENCH and Human Evaluation We report the above metrics of TASKBENCH to investigate the consistency with human evaluations. The results are shown in Table 8. We find that the average values for Kendall's τ and Spearman's ρ are 0.89 and 0.78, respectively. This indicates a very positive correlation between human evaluation and our TASKBENCH, which further validates the effectiveness of our proposed framework for dataset construction.

A.6 ANALYSIS

A.6.1 DIFFERENT NUMBER OF TOOLS

The greater the number of tool nodes the tool graphs contain, the more challenging it is for LLM to perform task automation. To make a clear understanding of the correlation between the number of nodes in the tool graph and the performance of LLMs in task automation, we conduct a detailed statistical analysis in Table 9. This analysis includes various metrics such as node-wise F1, node set accuracy, edge-wise F1, edge set accuracy, and graph accuracy, which measure the exact-match accuracy of the node set, edge set, and the entire graph, respectively.

Table 9: Task automation performance with different number of tools on GPT-4

# tool nodes	supports	node set accuracy	edge set accuracy	graph accuracy
1 (single)	2,059	96.16	-	96.16
2	278	86.33	84.53	84.53
3	1,313	67.93	60.70	60.39
4	1,280	64.29	75.62	54.37
5	731	54.03	70.53	41.58
6	290	50.34	39.31	39.31
7	151	49.66	36.42	36.42
8	60	35.00	25.00	25.00
9	55	38.18	21.81	21.81
10	64	39.06	31.25	31.25
overall	6,281	73.52	67.55	67.25

From Table 9, we observed that as the number of tools in the tool graphs increases, there is a clear downward trend in various performance metrics such as node set accuracy, edge set accuracy, and graph accuracy. This trend confirms that tool graphs with a higher number of tools present more complex challenges for LLMs in task automation. Specifically, the result shows a significant drop in performance metrics when moving from simpler graphs (1-2 tools) to more complex ones (3 or more tools). For instance, while single-node graphs achieve a high graph accuracy of 96.16%, this metric falls to 39.31% for graphs with 6 tools and further decreases to 25.00% for 8-node graphs.

This correlation between the number of tools and the difficulty of the test cases can be attributed to the increased complexity in understanding and processing more extensive and intricate links between tools. As the number of tools grows, LLMs must handle a larger set of possible dependencies, which significantly challenges their predictive and analytical capabilities. The results from this analysis underline the importance of continuous advancements in LLM capabilities to keep pace with the increasing complexity of tasks in various domains.

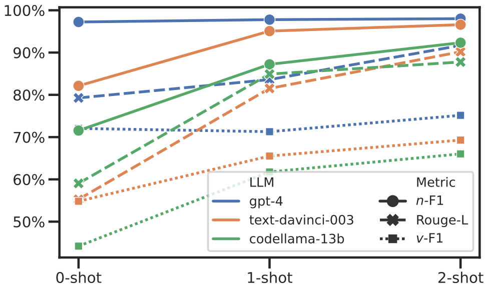


Figure 4: Performance in the few-shot setting for the Daily Life APIs domain. *R-L* for task decomposition; *n-F1* for tool selection; and *v-F1* for parameter prediction.

A.6.2 FEW-SHOT SETTING

In-context learning is a crucial capability for LLMs, that can improve the performance of LLMs by providing a few examples. In TASKBENCH, we also provide a fixed number of demonstrations in the designed prompt to advance the capability of LLMs in automation. Therefore, we also investigate the effect of the number of demonstrations in our setting. The results are reported in Table 10 and Figure 4. We can find that as the number of demonstrations increases, it can receive significant improvements of LLMs at different dimensions (e.g., task decomposition, tool selection, and parameter prediction). For example, codellama-13b with a 2-shot setting can obtain 20.78% and 21.82% improvements to the zero-shot setting in *n-F1* and *v-F1*. These results underscore the effect of the demonstrations in improving LLMs for task automation.

A.7 DETAIL COMPARISON OF LLMS

We present the performance of more open-source large language models (Team, 2023a; Li et al., 2023a; Team, 2023b) on TASKBENCH. The performance metrics for task decomposition, tool selection, and parameter prediction are shown in Table 11, Table 12, and Table 13, respectively.

Table 10: Performance in the few-shot setting. Rouge-L (*R-L*) reflects the performance on task decomposition; Node F1 (*n-F1*) and Edge F1 (*e-F1*) indicate the performance on tool selection; and Parameter Name F1 (*t-F1*) and Parameter Name & Value F1 (*v-F1*) indicate the performance on parameter prediction.

Shot	LLM	*R-L* ↑	*n-F1* ↑	*e-F1* ↑	*t-F1* ↑	*v-F1* ↑
0-shot	gpt-4	79.27	97.23	34.52	97.11	72.05
	gpt-3.5-turbo	48.72	86.45	51.77	83.85	56.34
	text-davinci-003	55.28	82.13	50.75	80.63	54.83
	codellama-13b	59.05	71.55	36.57	63.04	44.21
	wizardlm-13b	40.58	65.39	20.87	55.96	38.56
	vicuna-13b-v1.5	48.34	68.32	26.73	51.47	35.71
	nous-hermes-13b	36.58	50.44	13.96	36.32	25.07
1-shot	gpt-4	83.60	97.78	50.56	97.82	71.28
	gpt-3.5-turbo	76.16	90.87	51.14	90.18	62.31
	text-davinci-003	81.52	95.08	72.00	94.73	65.52
	codellama-13b	84.91	87.21	54.89	83.71	61.76
	vicuna-13b-v1.5	75.52	73.96	11.17	62.39	45.81
	nous-hermes-13b	73.04	72.69	2.76	63.77	46.48
	wizardlm-13b	75.96	67.93	12.26	53.59	39.08
2-shot	gpt-4	91.69	98.02	52.49	97.94	75.15
	gpt-3.5-turbo	89.07	96.03	39.72	95.28	69.04
	text-davinci-003	90.18	96.59	72.37	96.19	69.29
	codellama-13b	87.76	92.33	64.17	88.60	66.03
	nous-hermes-13b	78.88	82.42	43.55	74.68	54.39
	wizardlm-13b	80.20	79.25	33.51	70.69	52.20
	vicuna-13b-v1.5	79.67	79.84	37.50	71.45	51.82

Table 11: Evaluation for task decomposition. The scores for Rouge-1 (*R1*), Rouge-2 (*R2*), and BertScore F1 (*BsF*) for the generated step descriptions in comparison to the ground truth steps.

TASK DECOMPOSITION TASK - Step-by-step task decomposition									
LLM	**Hugging Face Tools**			**Multimedia Tools**			**Daily Life APIs**		
	R1 ↑	*R2* ↑	*BsF* ↑	*R1* ↑	*R2* ↑	*BsF* ↑	*R1* ↑	*R2* ↑	*BsF* ↑
gpt-4	52.42	30.38	90.12	60.84	40.08	91.19	85.07	72.36	96.91
gemini-pro	45.96	24.23	89.29	53.02	31.51	90.40	83.36	70.12	96.82
claude-2	44.21	21.12	88.71	48.85	23.59	89.22	82.26	69.88	96.64
text-davinci-003	36.68	17.61	87.03	49.23	27.97	89.21	68.27	50.30	93.59
gpt-3.5-turbo	42.99	21.58	88.47	49.66	28.51	89.54	58.53	39.90	91.29
codellama-13b	38.75	18.37	88.32	44.46	23.30	88.66	89.86	83.27	97.90
wizardlm-13b	34.47	15.38	87.38	35.87	17.55	87.29	82.02	72.43	96.36
vicuna-13b-v1.5	37.12	17.03	87.90	44.75	23.75	88.94	81.76	71.76	96.31
nous-hermes-13b	37.36	16.91	88.18	35.73	16.11	87.53	78.49	68.04	95.61
vicuna-33b-v1.3	33.52	14.75	86.73	31.27	13.37	86.17	54.96	39.71	91.40
codellama-7b	38.97	18.62	88.46	43.76	22.93	88.81	56.98	38.83	91.31
baichuan-13b-chat	19.93	5.97	83.85	20.41	3.77	83.31	49.43	27.25	88.32
llama-2-13b-chat	39.37	18.64	88.67	26.16	7.88	84.82	45.39	22.42	87.74
vicuna-7b-v1.5	27.17	10.02	85.61	39.46	19.83	88.53	40.26	21.19	87.27
mpt-7b-chat	33.21	12.73	87.23	30.94	11.90	86.08	44.54	20.98	87.17
llama-2-7b-chat	24.12	8.68	85.43	34.51	15.91	87.56	37.06	16.49	86.31
internlm-chat-7b	20.53	7.16	83.74	16.64	3.56	82.91	42.94	21.02	86.14
longchat-7b-v1.5	27.09	8.97	85.50	37.85	18.14	87.64	29.05	14.84	83.90

Table 12: Evaluation for tool selection. *n-F1* and *e-F1* for node and edge prediction. *NED* measures the normalized number of operations required to correct the prediction for chain structure.

TOOL SELECTION TASK - Predicts tools and their dependencies.

	LLM	Node	Chain			DAG		Overall	
		n-F1 ↑	*n-F1*↑	*e-F1* ↑	*NED* ↓	*n-F1*↑	*e-F1* ↑	*n-F1*↑	*e-F1* ↑
Hugging Face Tools	gpt-4	84.34	80.79	55.73	39.70	82.86	56.39	81.54	54.70
	gemini-pro	77.46	76.12	45.51	43.10	79.05	49.36	76.62	43.50
	claude-2	69.83	80.67	48.11	40.03	84.52	53.40	79.00	43.51
	gpt-3.5-turbo	56.91	72.63	39.92	46.52	73.79	38.55	69.49	33.36
	text-davinci-003	40.71	66.05	36.04	48.57	64.64	34.19	59.38	29.37
	codellama-13b	43.68	55.65	17.80	62.23	52.87	13.19	53.16	14.64
	baichuan-13b-chat	58.29	52.82	8.07	61.52	53.29	7.82	53.85	7.65
	nous-hermes-13b	58.66	52.39	9.01	62.48	51.99	6.33	53.62	8.29
	llama-2-13b-chat	43.59	49.87	8.22	64.99	49.60	9.11	48.47	7.30
	vicuna-13b-v1.5	51.74	50.37	8.40	66.83	52.46	9.06	50.82	7.28
	vicuna-33b-v1.3	42.73	43.39	5.80	71.95	45.09	5.61	43.40	4.82
	codellama-7b	18.81	47.70	8.52	63.55	45.20	7.17	37.59	5.35
	vicuna-7b-v1.5	36.20	44.79	3.24	69.40	43.94	2.00	42.87	2.76
	wizardlm-13b	54.69	54.50	2.22	60.55	52.93	0.92	54.40	2.05
	llama-2-7b-chat	14.89	32.61	0.71	81.01	31.47	1.38	27.30	0.74
	internlm-chat-7b	33.98	22.86	0.81	85.69	22.01	1.22	24.39	0.83
	longchat-7b-v1.5	44.97	49.11	0.52	65.74	48.41	1.04	48.18	0.56
	mpt-7b-chat	15.68	22.24	0.08	88.34	21.00	0.45	20.86	0.12
Multimedia Tools	gpt-4	97.13	89.70	69.29	28.93	92.32	71.64	90.90	69.27
	gemini-pro	73.61	82.65	55.50	35.62	85.29	57.80	81.54	52.07
	claude-2	66.16	83.95	59.22	33.41	82.98	54.28	80.94	53.01
	text-davinci-003	59.15	76.87	50.79	38.54	79.00	50.69	73.97	45.81
	gpt-3.5-turbo	53.55	76.81	50.30	39.05	78.65	49.52	72.83	44.02
	codellama-13b	43.70	66.89	28.77	46.35	68.68	28.79	62.78	24.61
	codellama-7b	40.43	56.15	16.90	54.36	57.55	16.71	53.29	14.76
	vicuna-13b-v1.5	66.64	59.18	16.49	54.17	61.40	13.95	60.61	14.78
	nous-hermes-13b	60.58	58.53	9.47	56.02	59.39	9.57	58.97	8.90
	wizardlm-13b	55.13	50.57	4.92	58.46	49.38	5.52	51.24	4.82
	baichuan-13b-chat	45.59	41.96	4.95	64.28	42.05	8.46	42.51	5.19
	longchat-7b-v1.5	43.54	42.72	4.25	67.09	44.83	5.30	43.08	3.95
	vicuna-7b-v1.5	36.22	48.29	4.79	63.49	48.26	4.09	46.06	4.26
	llama-2-13b-chat	38.02	45.14	1.62	65.29	45.95	2.11	43.87	1.63
	llama-2-7b-chat	16.49	30.00	0.94	76.13	28.81	1.23	26.47	0.91
	internlm-chat-7b	36.39	22.21	1.17	84.65	22.53	1.03	23.60	1.14
	mpt-7b-chat	11.85	7.99	0.12	95.81	9.40	0.66	8.68	0.18
	vicuna-33b-v1.3	27.53	2.89	0.02	97.56	1.82	0.00	6.40	0.01
Daily Life APIs	gpt-4	95.97	97.06	83.47	38.69	96.41	42.01	96.91	80.53
	claude-2	79.57	95.36	80.68	39.93	93.85	41.04	93.52	75.31
	gemini-pro	76.15	92.79	64.58	41.64	89.68	28.42	90.75	59.45
	gpt-3.5-turbo	52.18	90.80	70.66	43.50	86.94	30.85	85.37	60.67
	text-davinci-003	68.49	82.15	60.12	47.14	76.81	24.54	80.42	54.90
	codellama-13b	89.75	87.80	65.92	44.42	83.61	27.47	87.73	63.16
	codellama-7b	40.19	62.00	31.11	59.14	58.19	13.35	59.33	27.23
	llama-2-13b-chat	34.11	57.61	20.13	67.06	56.18	8.42	55.77	17.02
	vicuna-33b-v1.3	30.25	57.34	19.72	62.65	49.13	8.52	52.49	16.37
	vicuna-7b-v1.5	46.51	54.01	17.43	65.38	51.68	10.68	52.73	14.23
	longchat-7b-v1.5	34.20	49.91	18.17	69.96	53.53	11.93	47.26	14.44
	wizardlm-13b	92.27	65.74	14.51	55.80	63.80	9.20	69.34	14.18
	vicuna-13b-v1.5	90.59	73.74	13.24	51.43	67.92	5.62	75.67	12.48
	baichuan-13b-chat	52.50	52.60	11.59	69.27	52.08	6.53	52.55	10.61
	internlm-chat-7b	33.08	29.28	7.06	86.26	22.22	3.62	29.14	6.63
	llama-2-7b-chat	20.11	31.68	5.40	83.87	30.88	2.80	30.17	4.27
	nous-hermes-13b	92.50	71.17	3.55	53.47	70.65	2.86	73.45	3.50
	mpt-7b-chat	14.75	16.24	2.34	93.28	16.18	1.40	15.95	1.69

Table 13: Evaluation for tool parameter prediction. Parameter Name F1 *(t-F1)* evaluates (task, parameter name) pairs, while *v-F1* assesses (task, parameter name, parameter value) triples.

TOOL PARAMETER PREDICTION TASK - Predicts parameters for the tool execution.									
	LLM	**Node**		**Chain**		**DAG**		**Overall**	
		t-F1↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑	*t-F1* ↑	*v-F1*↑
Hugging Face Tools	gpt-4	80.05	74.10	76.66	58.15	78.24	60.03	77.31	60.86
	gemini-pro	67.63	56.54	66.60	46.35	70.41	50.56	67.12	48.54
	claude-2	48.07	32.14	66.35	45.57	68.59	48.19	63.00	43.08
	text-davinci-003	38.51	27.43	56.90	38.76	57.03	38.90	52.53	36.04
	gpt-3.5-turbo	37.70	19.81	60.96	41.15	61.33	42.89	55.88	36.32
	codellama-13b	20.09	12.58	36.40	21.31	33.43	20.48	32.06	18.87
	nous-hermes-13b	46.38	31.06	35.55	13.81	33.06	13.69	37.51	17.66
	wizardlm-13b	43.97	25.90	37.34	12.48	38.43	13.79	38.76	15.35
	llama-2-13b-chat	29.80	20.54	32.14	13.57	32.16	15.23	31.61	15.38
	baichuan-13b-chat	46.18	29.46	30.29	9.55	30.10	10.37	33.17	13.53
	longchat-7b-v1.5	34.94	19.37	33.07	11.39	34.06	13.75	33.57	13.94
	vicuna-13b-v1.5	25.71	13.11	28.99	11.14	30.04	13.60	28.34	11.85
	vicuna-7b-v1.5	20.82	12.56	25.85	10.10	26.09	10.94	24.65	10.81
	vicuna-33b-v1.3	20.75	11.89	23.23	9.28	23.97	10.89	22.71	10.07
	codellama-7b	13.31	4.48	27.47	11.97	24.94	12.36	22.50	9.20
	internlm-chat-7b	20.52	14.08	14.29	4.76	14.44	5.62	15.41	6.64
	llama-2-7b-chat	7.61	2.46	15.53	2.81	15.42	4.15	13.05	2.79
	mpt-7b-chat	6.30	1.72	10.80	1.84	10.35	2.08	9.61	1.83
Multimedia Tools	gpt-4	95.64	87.12	85.60	69.83	87.57	72.79	87.06	72.31
	gemini-pro	62.21	50.48	72.99	55.21	76.13	58.79	71.67	54.82
	claude-2	53.81	24.02	75.60	58.12	72.41	52.43	71.63	51.58
	gpt-3.5-turbo	44.94	11.96	70.53	47.76	71.82	47.95	65.91	40.80
	text-davinci-003	60.30	20.78	69.91	44.76	71.91	45.76	68.48	40.70
	codellama-13b	32.01	16.10	52.30	32.51	53.08	33.79	48.19	29.13
	codellama-7b	31.79	23.10	39.42	24.50	40.52	26.98	38.04	24.45
	vicuna-13b-v1.5	52.72	35.55	39.31	21.00	40.05	21.40	41.62	23.62
	nous-hermes-13b	50.11	37.80	41.98	17.89	43.99	20.04	43.60	21.69
	wizardlm-13b	49.79	33.59	36.88	14.87	36.61	18.68	39.10	18.74
	vicuna-7b-v1.5	28.79	17.79	29.73	12.48	31.38	14.12	29.72	13.74
	longchat-7b-v1.5	31.06	21.12	26.97	11.07	28.43	14.16	27.89	13.41
	baichuan-13b-chat	40.41	27.87	25.80	8.50	25.87	10.13	28.04	11.77
	llama-2-13b-chat	28.49	17.01	30.26	9.66	31.00	11.35	29.99	11.32
	internlm-chat-7b	24.01	16.04	12.45	4.81	13.21	5.54	13.75	6.09
	llama-2-7b-chat	14.00	7.03	19.73	5.38	19.20	5.78	18.27	5.84
	mpt-7b-chat	4.11	2.15	2.84	0.64	3.78	1.20	3.19	1.02
	vicuna-33b-v1.3	9.51	4.71	0.90	0.16	0.37	0.12	2.47	1.09
Daily Life APIs	gpt-4	95.83	76.21	97.23	70.67	95.95	69.65	97.02	71.14
	claude-2	78.12	59.43	94.72	65.30	91.83	66.39	92.71	64.72
	gemini-pro	69.88	45.41	91.66	57.93	88.50	53.91	88.95	56.22
	gpt-3.5-turbo	43.81	28.77	89.21	61.11	83.88	56.13	81.97	55.66
	text-davinci-003	61.68	45.53	80.68	54.54	76.51	51.91	78.37	53.40
	codellama-13b	86.34	71.20	84.31	61.51	80.42	60.18	84.26	62.38
	nous-hermes-13b	79.69	63.29	62.64	45.32	63.26	45.74	64.47	47.22
	vicuna-13b-v1.5	83.63	67.71	61.80	44.54	57.14	41.72	64.27	47.31
	wizardlm-13b	89.27	72.96	50.68	36.48	49.03	35.75	55.00	40.53
	codellama-7b	31.62	21.16	56.33	37.20	52.56	33.46	52.99	34.81
	vicuna-33b-v1.3	21.13	17.20	44.61	32.72	35.69	27.01	39.95	29.64
	vicuna-7b-v1.5	27.71	19.81	38.25	25.82	37.16	24.65	36.30	24.67
	baichuan-13b-chat	32.47	21.72	38.31	24.24	36.84	21.84	37.48	23.77
	llama-2-13b-chat	10.39	7.32	38.89	25.37	36.43	23.40	35.11	22.94
	longchat-7b-v1.5	14.99	12.11	28.37	19.60	31.25	22.22	25.73	18.18
	internlm-chat-7b	18.67	15.22	19.56	13.50	14.48	10.80	19.21	13.48
	llama-2-7b-chat	6.60	4.21	16.85	10.53	16.95	10.46	14.94	9.34
	mpt-7b-chat	2.80	2.03	6.45	4.14	5.74	3.25	5.34	3.45

Table 14: Statistics for the TaskBench. We report the number of nodes and links of the tool graphs. "# Avg. Nodes" and "# Avg. Links" stands for the average number of nodes and links involved in one sample. We also report the sample number and average request length for the datasets.

Statistic		**Hugging Face Tools**	**Multimedia Tools**	**Daily Life APIs**
# Nodes of Tool Graph		23	40	40
# Links of Tool Graph		225	449	1,560
# Avg. Nodes		3.47	3.68	3.82
# Avg. Links		2.46	2.68	2.8
# Samples		12,217	8,904	7,150
- Node / Chain / DAG		3,270 / 4,302 / 4,645	2,117 / 3,145 / 3,642	1,277 / 2,716 / 3,157
Avg. Request Length		41.21	39.15	38.64
- Node / Chain / DAG		28.42 / 45.72 / 46.04	24.71 / 43.55 / 43.73	12.36 / 44.49 / 44.23
self-critic	Both critics	8,456 (69.22%)	6,281 (70.54%)	5,432 (75.97%)
	LLM-based critic	9,042 (74.01%)	6,959 (78.16%)	5,694 (79.63%)
	Rule-based critic	10,289 (84.22%)	7,363 (82.69%)	6,271 (87.70%)
Human Verification		7,546 (61.76%)	5,584 (62.71%)	4,320 (60.42%)

A.8 Details about Back-Instruct and TaskBench

Table 14 reports the statistical information of the tool graph and our constructed TaskBench datasets across three domains. Notably, it is evident that the two critics we introduced play a crucial role in improving data quality. The rule-based and LLM-based critics respectively filter out an average of 15.13% and 22.73% of the samples. In addition, we invited human experts to revise and filter the data. And finally, we obtained 61.76%, 62.71%, and 60.42% of the aligned samples for the three datasets, respectively.

We utilize the following prompt template for the "Back-Instruct" Data Engine. Each sample is generated through a single prompting. We assign "instruction", "tool invocation graph", and "self-critics" to specific fields in the prompt, and then populate the relevant fields to complete the data generation in a single prompt.

```
Given a tool graph where tools serve as nodes and invoking chains between
 tools act as edges, the following tools (nodes) are available with their
 respective descriptions and input/output types:
{NODES}

These tools can be connected as follows, where the directed edges
represent invoking chains between tools:
{EDGES}

Based on the above tool graph, please skillfully generate the
corresponding task steps, user request, and tool invoking graph.

Requirements:
1. The generated user request should be clear, self-contained (with user-
specified text, image, video, audio, and content included within the
request) and practical (designed to help users solve a tangible problem).
 The task steps must strictly adhere to the tool graph (nodes and edges)
and be reasonable. The tool invoking graph must align with both the task
steps and the provided tool graph.
2. The user request should be decomposable into task steps that the tool
invoking graph can solve.
3. Each task step must correspond to a tool node in both the tool graph
and the tool invoking graph. The number of task steps must equal the
number of nodes. Each tool node can only be used once.
4. If the user request requires image/audio/video resources, use files
named 'example.[jpg/mp4/wav/png]'.
5. The dependencies among task steps must be consistent with the edges of
 both the tool graph and the tool invoking graph.
```

```
Now, please generate your result (with random seed {seed}) in a strict
JSON format:

{
"task_steps": [step description for one or more steps],
"user_request": "your high-quality and self-contained synthesized request
",
"invoking_graph": {
    "nodes": [
        {
            "id": "tool name",
            "input": [either user-specified text or resource file '
            example.[jpg/mp4/wav/png]' from the user request, or the name
             of the dependent tool whose output this node requires]
        }
    ],
    "links": [{"source": "tool name i", "target": "tool name j"}]
    },
"check_by_teacher": "This field is filled by your strict and well-trained
 teacher, minor mistakes are complete intolerable to him. He evaluated
whether your synthesized user request, tool invoking graph are valid and
whether they are aligned with the given tool graph (strictly checked step
 by step according to the above requirements). Some comments from him
place here (start with 'Let me check your result step by step, and
evaluate the 'Executable' and 'Correct' of the tool invoking graph (
Executable means that the tool invoking graph executed successfully,
regardless of alignment with the given tool graph. While Correct implies
that the tool invoking graph are not only 'Executable' but also strictly
consistent (with strictly same nodes and same edges) with the given tool
graph). After carefully evaluating, found some mistakes:' and end with a
conclusion: 'Conclusion: Executable: no/yes, Correct: no/yes'.)
}:
```

A.9 PROMPT FOR INFERENCE

For a fair and standardized evaluation, we provide prompt templates for inference.

```
# Tools List #
FOR tool in {tool_list}:
    {tool["id"]: {tool["description"]}}
    Parameters: {tool["parameter"]}
END FOR

# GOAL #:
Based on the above tools, I want you to generate task steps and a task
graph (tool invocation graph, including nodes and edges) to address the #
 USER REQUEST #. The format must be in strict JSON format, like:
{
    "task_steps": [step description for one or more steps],
    "task_nodes": [{
        "task": "tool name must be from # TOOL LIST #",          "
        arguments": [a concise list of arguments for the tool. This can
        be original text, a user-mentioned filename, or the tag '<node-j
        >' (starting from 0) to refer to the output of the j-th node.]
        }]
    "task_links": [{"source": "task name i", "target": "task name j"}],
}.

# REQUIREMENTS #:
1. The generated task steps and task nodes must resolve the given user
request # USER REQUEST # perfectly. Task names must be selected from #
TASK LIST #.
2. The task steps should align strictly with the task nodes, and the
number of task steps should be the same as the task nodes.
```

```
3. The dependencies among task steps should be consistent with the
argument dependencies of the task nodes.
4. The tool arguments should align with the parameter field of # TASK
LIST #.

# EXAMPLE #:
FOR demo IN {demos}:
# USER REQUEST #:
{demo["user_request"]}
# RESULT #:
{(demo["result"])}
END FOR

# USER REQUEST #:
{{user_request}}
Now, please generate your result in strict JSON format:
# RESULT #:
```

A.10 Tools in the Tool Graph

We show some of the tools used in the construction of the tool graph, including the tool name, tool description and parameters of the tool. In the Daily Life APIs domain, we resorted to manual construction because of the scarcity of publicly available APIs. We crafted 40 APIs tailored to common daily life activities such as shopping, education, and travel. Our focus is solely on producing the API documentation without implementing the actual functionality. Some of the tools on the Hugging Face tools, Multimedia tools and Daily Life APIs domains are shown in Table 15, Table 16, and Table 17, respectively.

In order to visualize the complete tool graph we constructed, we take the Multimedia domain as an example to render the tool graph with resource dependencies. As shown in Figure 6, nodes in the graph denote tools, and directed edges indicate that the output type of the source tool matches the input type of the target tool.

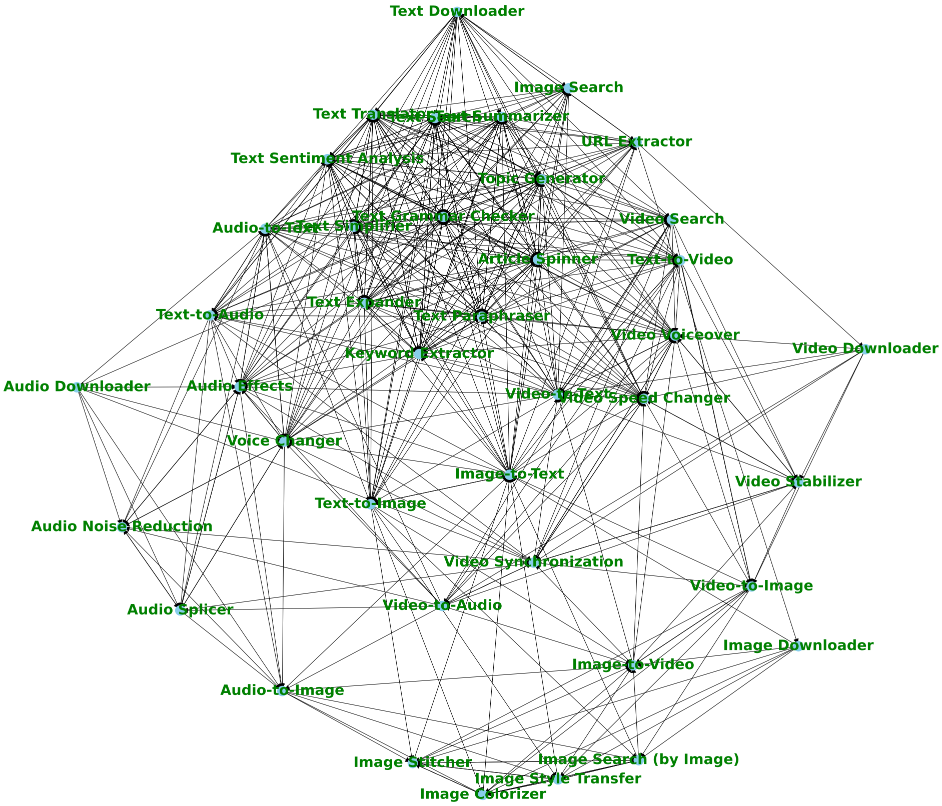


Figure 5: Constructed tool graph with resource dependencies on the Multimedia Tools domain.

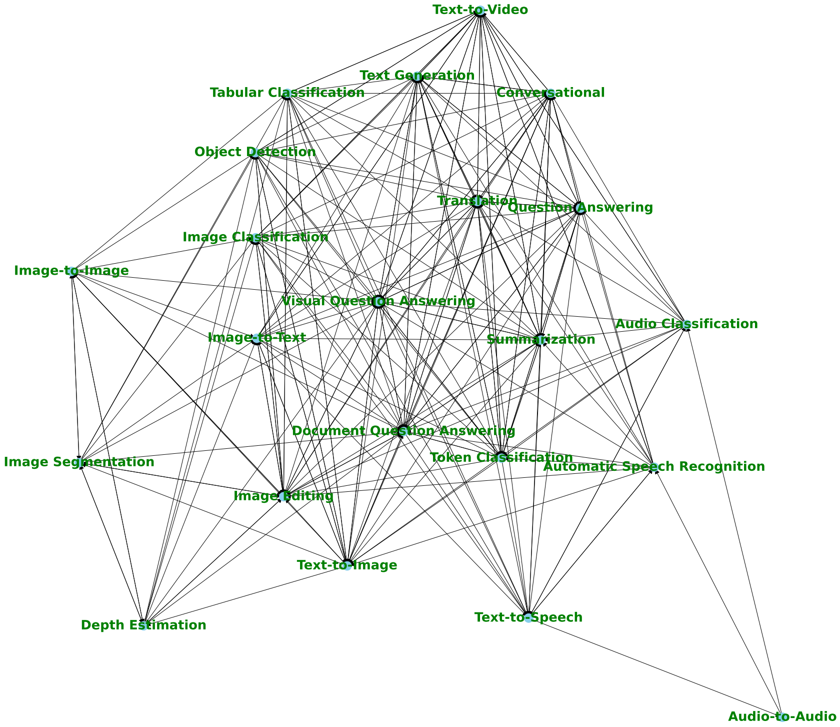


Figure 6: Constructed tool graph with resource dependencies on the Hugging Face Tools domain.

Table 15: Hugging Face tools and their descriptions

Name	Description	Parameters
Translation	Translation is the task of converting text from one language to another.	[’text’]
Summarization	Summarization is the task of producing a shorter version of a document while preserving its important information. Some models can extract text from the original input, while other models can generate entirely new text.	[’text’]
Question Answering	Question Answering models can retrieve the answer to a question from a given text, which is useful for searching for an answer in a document.	[’text’, ’text’]
Text Generation	Generating text is the task of producing new text. These models can, for example, fill in incomplete text or paraphrase.	[’text’]
Object Detection	Object Detection models allow users to identify objects of certain defined classes. Object detection models receive an image as input and output the images with bounding boxes and labels on detected objects.	[’image’]
Image Classification	Image classification is the task of assigning a label or class to an entire image. Images are expected to have only one class for each image. Image classification models take an image as input and return a prediction about which class the image belongs to.	[’image’]
Image-to-Image	Image-to-image is the task of transforming a source image to match the characteristics of a target image or a target image domain. Any image manipulation and enhancement is possible with image to image models.	[’image’]
Image-to-Text	Image to text models output a text from a given image. Image captioning or optical character recognition can be considered as the most common applications of image to text.	[’image’]
Text-to-Image	Generates images from input text. These models can be used to generate images based on text prompts.	[’text’]
Text-to-Video	Generates videos from input text. These models can be used to generate videos based on text prompts.	[’text’]
Visual Question Answering	Visual Question Answering is the task of answering questions based on an image.	[’image’, ’text’]
Image Segmentation	Image Segmentation divides an image into segments where each pixel in the image is mapped to an object. This task has multiple variants such as instance segmentation, panoptic segmentation and semantic segmentation.	[’image’]
Depth Estimation	Depth estimation is the task of predicting depth of the objects present in an image.	[’image’]
Text-to-Speech	Text-to-Speech (TTS) is the task of generating natural sounding speech given text input. TTS models can be extended to have a single model that generates speech for multiple speakers and multiple languages.	[’text’]
Automatic Speech Recognition	Automatic Speech Recognition (ASR), also known as Speech to Text (STT), is the task of transcribing a given audio to text. It has many applications, such as voice user interfaces.	[’audio’]
Audio-to-Audio	Audio-to-Audio is a family of tasks in which the input is an audio and the output is one or multiple generated audios. Some example tasks are speech enhancement and source separation.	[’audio’]
Audio Classification	Audio classification is the task of assigning a label or class to a given audio. It can be used for recognizing which command a user is giving or the emotion of a statement, as well as identifying a speaker.	[’audio’]
Image Editing	Image editing is the task of modifying an image to match a given text description. It can be used to modify the attributes of an image, such as the color of an object or the background.	[’text’, ’image’]

Table 16: Multimedia tools and their descriptions

Name	Description	Parameters
Image Downloader	Downloads an image from a given URL.	[’url’]
Video Downloader	Downloads a video from a given URL.	[’url’]
Audio Downloader	Downloads an audio file from a given URL.	[’url’]
Text Downloader	Downloads the text content from a given URL.	[’url’]
Text Search	Searches for a specific text or keyword on the internet.	[’text’]
Image Search	Searches for images on the internet based on a given query.	[’text’]
URL Extractor	Extracts URL from text	[’text’]
Video Search	Searches for videos on the internet based on a given query.	[’text’]
Text-to-Video	Generates a video based on a given text description.	[’text’]
Text-to-Audio	Generates an audio file based on a given text description.	[’text’]
Image-to-Text	Extracts text from an input image using Optical Character Recognition (OCR).	[’image’]
Audio-to-Text	Transcribes speech from an audio file into text.	[’audio’]
Video-to-Text	Transcribes speech from a video file into text.	[’video’]
Audio Noise Reduction	Reduces background noise or unwanted sounds from a given audio file.	[’audio’]
Audio Effects	Applies various audio effects to a given audio file according to human instruction, such as reverb, chorus, or equalization.	[’audio’, ’text’]
Audio Splicer	Combines two audio files into a single output file.	[’audio’, ’audio’]
Voice Changer	Modifies the characteristics of a recorded voice according to human instruction, such as tone, pitch, or gender.	[’audio’, ’text’]
Text Summarizer	Summarizes a given text into a shorter version while retaining the main points.	[’text’]
Text Translator	Translates a given text from one language to english.	[’text’]
Text Sentiment Analysis	Analyzes the sentiment of a given text, identifying if it is positive, negative, or neutral.	[’text’]
Text Grammar Checker	Checks a given text for grammatical errors and suggests corrections.	[’text’]
Text Simplifier	Rewrites a given text in a simpler and more understandable manner.	[’text’]
Keyword Extractor	Extracts the most important keywords and phrases from a given text.	[’text’]
Text Paraphraser	Rewrites a given text using different words while maintaining its original meaning.	[’text’]
Topic Generator	Generates a list of relevant topics or ideas based on a given input.	[’text’]
Audio-to-Image	Generates an image that visually represents a given audio, such as a waveform or spectrogram.	[’audio’]
Video-to-Audio	Extracts the audio track from a given video file.	[’video’]
Video-to-Image	Extracts a still image from a given video.	[’video’]
Image Stitcher	Stitches together two input images to create a panorama or collage.	[’image’, ’image’]
Image Colorizer	Adds color to a black and white input image using deep learning techniques.	[’image’]
Video Stabilizer	Stabilizes a shaky input video to produce a smoother output video.	[’video’]
Video Speed Changer	Adjusts the playback speed of a given video according to human instruction, either speeding it up or slowing it down.	[’video’, ’text’]
Video Synchronization	Synchronizes the timing of an existing voiceover or audio file with the visuals of a given video.	[’video’, ’audio’]

Table 17: Daily Life APIs and their descriptions

API Name	API Description	Parameter Names
get_news_for_topic	Get the news for a specific topic	['topic']
stock_operation	Do a specific operation on a specific stock	['stock', 'operation']
book_flight	Book a flight for a specific date, from a specific location to a specific destination	['date', 'from', 'to']
book_hotel	Book a specific hotel for a specific date	['date', 'name']
book_car	Book a car for a specific date, in a specific location	['date', 'location']
online_shopping	Buy a product from a specific website	['website', 'product']
send_email	Send an email to a specific email address	['email_address', 'content']
send_sms	Send an sms to a specific phone number	['phone_number', 'content']
share_by_social_network	Share a specific content by a specific social network	['content', 'social_network']
book_restaurant	Book a specific restaurant for a specific date	['date', 'name']
search_by_engine	Search a specific query by a specific search engine	['query', 'engine']
apply_for_job	Apply for a specific job	['job']
see_doctor_online	See a specific doctor for a specific disease	['disease', 'doctor']
consult_lawyer_online	Consult a specific lawyer for a specific legal issue	['issue', 'lawyer']
enroll_in_course	Enroll in a specific course at a specific university	['course', 'university']
buy_insurance	Buy a specific insurance from a specific insurance company	['insurance', 'company']
online_banking	Do a specific banking operation online at a specific bank	['instruction', 'bank']
daily_bill_payment	Pay a specific bill	['bill']
sell_item_online	Sell a specific item at a specific online store	['item', 'store']
do_tax_return	Do the tax return for a specific year	['year']
apply_for_passport	Apply for a passport	['country']
pay_for_credit_card	Pay for a specific credit card	['credit_card']
auto_housework_by_robot	Let a robot do a housework by following a specific instruction	['instruction']
auto_driving_to_destination	Let a car drive to a specific destination	['destination']
deliver_package	Deliver a specific package to a specific destination	['package', 'destination']
order_food_delivery	Order a specific food to be delivered to a specific location at a specific platform	['food', 'location', 'platform']
order_taxi	Order a taxi to a specific location at a specific platform	['location', 'platform']
play_music_by_title	Play a specific music by a specific title	['title']