Budget-Constrained Tool Learning with Planning

Yuanhang Zheng[1], **Peng Li**[2*], **Ming Yan**[3], **Ji Zhang**[3], **Fei Huang**[3] **and Yang Liu**[1,2,4*]
[1]Dept. of Comp. Sci. & Tech., Institute for AI, Tsinghua University, Beijing, China
[2]Institute for AI Industry Research (AIR), Tsinghua University, Beijing, China
[3]Institute of Intelligent Computing, Alibaba Group
[4]Jiangsu Collaborative Innovation Center for Language Competence, Jiangsu, China

Abstract

Despite intensive efforts devoted to tool learning, the problem of budget-constrained tool learning, which focuses on resolving user queries within a specific budget constraint, has been widely overlooked. This paper proposes a novel method for budget-constrained tool learning. Our approach involves creating a preferable plan under the budget constraint before utilizing the tools. This plan outlines the feasible tools and the maximum number of times they can be employed, offering a comprehensive overview of the tool learning process for large language models. This allows them to allocate the budget from a broader perspective. To devise the plan without incurring significant extra costs, we suggest initially estimating the usefulness of the candidate tools based on past experience. Subsequently, we employ dynamic programming to formulate the plan. Experimental results demonstrate that our method can be integrated with various tool learning methods, significantly enhancing their effectiveness under strict budget constraints.[1]

Method	**Budget Constraint**					
	$+\infty$			20		
	PR↑	PBC↑	AC↓	PR↑	PBC↑	AC↓
ReAct	44.0	44.0	15.4	**44.0**	34.1	15.4
+BTP (*Ours*)	**46.3**	**46.3**	**9.0**	43.7	**43.7**	**6.9**
DFSDT	63.8	63.8	78.3	63.8	28.8	78.3
+BTP (*Ours*)	**66.1**	**66.1**	**12.5**	**64.5**	**64.5**	**9.2**
ToT-DFS	61.6	61.6	51.4	61.6	10.2	51.4
+BTP (*Ours*)	**65.0**	**65.0**	**15.8**	**64.1**	**64.1**	**10.8**

Table 1: Comparison of **Pass Rate (PR)**, **Pass rate under Budget Constraint (PBC)** and **Average Cost (AC)** on the ToolBench (Qin et al., 2023b) dataset. Our proposed **B**udget-Constrained **T**ool Learning with **P**lanning (BTP) reduces the cost of tool learning and reaches competitive Pass Rate, significantly improving the performance under a strict budget constraint.

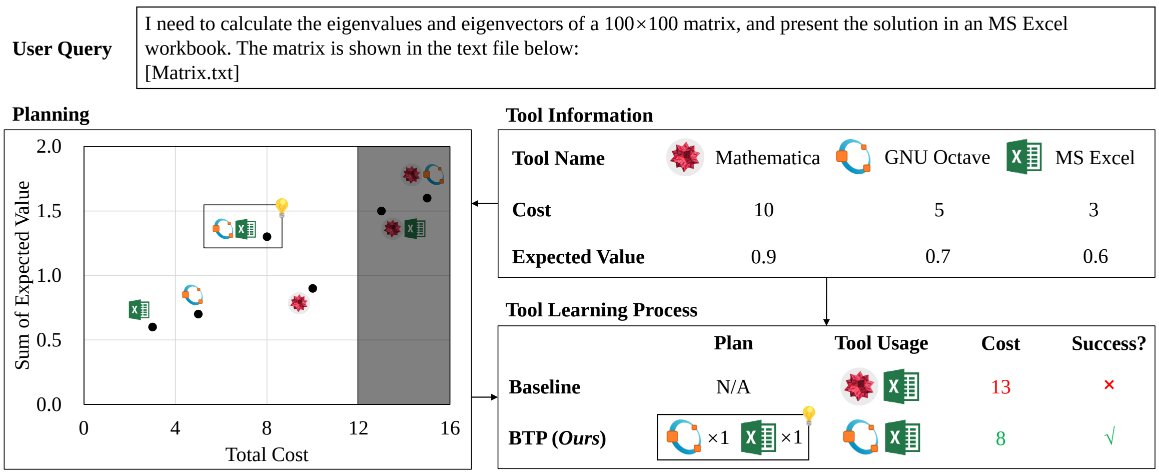
1 Introduction

Tool learning (Schick et al., 2023; Yao et al., 2023b; Qin et al., 2023b), which uses external tools to extend the capability of large language models (LLMs), has achieved remarkable results on various types of tasks. For example, with the aid of external tools, LLMs may solve difficult mathematical problems with higher accuracy (Cobbe et al., 2021; Gao et al., 2023b), handle multimodal information (Shen et al., 2023; Lu et al., 2023), or interact with real-world applications (Song et al., 2023; Gur et al., 2023). To better resolve more complex user queries, previous studies have proposed different tool learning methods which allow the LLM to use multiple tools (Chen et al., 2023; Yao et al., 2023b; Paranjape et al., 2023; Shen et al., 2023; Qin et al., 2023b).

Despite their effectiveness, existing methods often overlook a critical aspect: utilizing tools incurs expenses, such as money and time, and users may have (implicit) budget constraints in real-world scenarios. Without meticulous management, the costs can rapidly exceed the acceptable threshold of a user. In scenarios where the expenses overshadow the benefits, even if the tool successfully addresses the intended problem, users may still perceive the solution as unsatisfactory due to the disproportionate incurred costs. Table 1 presents preliminary results for three strong tool learning approaches, ReAct (Yao et al., 2023b), DFSDT (Qin et al., 2023b), and ToT-DFS (Yao et al., 2023a), evaluated on the ToolBench (Qin et al., 2023b) dataset, with success defined strictly in terms of problem resolution within budget constraints. It is evident that all three methods experience significant declines in performance as the budget constraint tightens from unlimited to 20. Consequently, we argue that more

[*] Corresponding authors: P.Li (`lipeng@air.tsinghua.edu.cn`) and Y.Liu (`liuyang2011@tsinghua.edu.cn`).

[1]Code can be found at `https://github.com/THUNLP-MT/BTP`.



User Query: I need to calculate the eigenvalues and eigenvectors of a 100×100 matrix, and present the solution in an MS Excel workbook. The matrix is shown in the text file below:
[Matrix.txt]

Tool Name	Mathematica	GNU Octave	MS Excel
Cost	10	5	3
Expected Value	0.9	0.7	0.6

	Plan	Tool Usage	Cost	Success?
Baseline	N/A		13	×
BTP (*Ours*)	×1 ×1		8	√

Figure 1: Example of budget-constrained tool learning. In this example, the baseline fails to resolve the user query within the budget constraint. Our proposed BTP makes a preferable plan of tool usage before using the tools, which may help resolve the user query under the budget constraint. "Expected Value" measures how valuable the candidate tool is for resolving the given query, which is estimated based on the past experience of tool learning. The shaded area in the "Planning" part means that the candidate plans in the area exceed the budget constraint.

efforts are worth devoting to the problem of *budget-constrained tool learning*, which aims at resolving user queries within a given budget constraint.

One fundamental challenge in budget-constrained tool learning is the allocation of the budget. However, determining the optimal budget allocation is not straightforward without a comprehensive understanding of the entire tool learning process. Taking Figure 1 as an example, the user query may be addressed using either Mathematica and MS Excel, or GNU Octave and MS Excel. While Mathematica, a commercial software, may provide a more precise solution than the open-source GNU Octave, we may incorrectly choose Mathematica without knowing the additional requirement for MS Excel. Unfortunately, acquiring such a comprehensive view is difficult. A simplistic approach is employing trial-and-error methods. However, each attempt introduces additional costs, further complicating adherence to the budget constraint.

To this end, we propose **B**udget-Constrained **T**ool Learning with **P**lanning (BTP), a novel method for budget-constrained tool learning, which finds a preferable plan under the budget constraint before using the tools (Figure 1). First, for each candidate tool, we estimate its expected value and frequency constraint based on the past experience. The expected value measures how valuable the tool is for resolving the given query, while the frequency constraint limits the maximum number of times it can be used. Then, we try to find a plan that maximizes the sum of the expected value under the budget constraint. A plan specifies the viable tools and how many times each of them can be used. We regard this as a knapsack problem and use dynamic programming to resolve it. Finally, we apply the plan during the tool learning process. As shown in Table 1, our proposed method can be combined with different tool learning methods and improve their performance under a strict budget constraint.

2 Methodology

In this section, we first introduce a formulation of budget-constrained tool learning (Section 2.1). Then, we describe our proposed BTP in detail. Before using the tools, we try to find a preferable plan under the budget constraint (Section 2.2). Then, we apply the plan to the process of tool learning (Section 2.3). Moreover, we introduce a blacklist mechanism to further reduce the cost of tool learning (Section 2.4).

2.1 Budget-Constrained Tool Learning

Existing tool learning methods can be roughly divided into two lines. The first line finetunes the LLMs to make them capable of using tools without the documentation of the tools (Schick et al., 2023; Hao et al., 2023), while the second line lets the LLM select tools according to the documentation of the candidate tools (Shen et al., 2023; Hsieh et al., 2023; Qin et al., 2023b). In this paper, we mainly focus on the latter line, since it can better support closed-source LLMs, including those which do not support fine-tuning. Typically, the tool learning process following this line is as follows: First, a set of n candidate tools $\mathcal{T} = \{t_1, \ldots, t_i, \ldots, t_n\}$ are retrieved according to the user query using a retrieval model. Then an LLM is leveraged to select and invoke tools iteratively until a stop condition is met. Assume that this process runs for N times, we denote the tools invoked as $\mathcal{T}_\text{I} = \{t_{a_1}, \cdots, t_{a_j}, \cdots, t_{a_N}\}$. Note that a tool may be invoked multiple times. Finally, the LLM produces the final output based on the user query and the results returned by the executed tools.

To conduct budget-constrained tool learning, we first propose a simplified and approximate but useful mathematical formulation for it. Formally, for any integer $i \in [1, n]$, we use c_i to represent the cost of using the tool t_i once.[2] In reality, this cost may be the estimated total cost spent on the tool in different aspects, including the money spent on using the tool and the LLM, as well as the consumed time for using the tool and the LLM:

$$c_i = c_{i,\text{tool}} + c_{\text{LLM}} + T_i, \tag{1}$$

where $c_{i,\text{tool}}$ represents the estimated cost of using the tool t_i itself, c_{LLM} is the estimated cost of using the API of the LLM once, and T_i is the estimated consumed time for using the tool and the LLM. We implicitly assume that all costs involved can be quantified using the same unit of measurement. For example, we can apply a conversion function to translate time costs into monetary terms.

Moreover, we use c_s to represent the extra cost produced by the other factors such as the system prompt and the user prompt, and B to represent the budget constraint. Then, *the target of budget-constrained tool learning is to resolve a user query under the budget constraint:*

$$c_s + \sum_j c_{a_j} \leq B. \tag{2}$$

In practice, c_s and c_i are influenced by numerous factors, complicating the design of a feasible algorithm. To simplify the problem and make it manageable, we introduce an assumption that both c_s and c_i can be estimated in advance. The assumption does not significantly limit the generality of our approach. On the one hand, both of them can be estimated using their historical average values. On the other hand, c_s can be more precisely estimated based on the price of the API of the LLM, the lengths of the system prompt and the user prompt for the current user query.

Currently, we do not take the cost arising from the external environments into consideration. For example, when the LLM uses an online shopping tool to buy a book for the user, the price of the book also consumes the overall budget and is difficult to be estimated in advance. Addressing this aspect is designated for future research.

2.2 Tool Usage Planning

In this work, we introduce a tool usage plan to offer an estimated overview of the tools likely to be utilized by the LLM, aiding in more effective budget allocation. The plan outlines the set of available tools and the maximum number of times each tool can be used.

However, it is difficult to precisely calculate how many times each candidate tool should be used for resolving a given user query. On the one hand, it is not trivial to estimate whether a candidate tool can provide valuable information for resolving the user query without using the tool. On the other hand, calculating the costs of every possible plan of tool usage for a given user query is unacceptable in terms of time complexity. Thus, we propose an approximate planning method for budget-constrained tool learning. In this planning method, we estimate whether the candidate tools can provide valuable information based on the past experience, and use dynamic programming to estimate how many times each candidate tool should be used.

Specifically, we first estimate the expected value of each candidate tool, which measures how valuable the results returned by the candidate tool are. In this work, we leverage the past experience of tool learning to estimate the expected value. Formally, the past experience $\mathcal{E} = \{U_1, \ldots, U_k, \ldots, U_m\}$ consists of a number of tool usages U_k which happened in the past. Each tool usage $U_k = \langle q, d(t), p, r\rangle$ consists of a user query q, the documentation $d(t)$ of the used tool t, the input parameters p and the returned result r. We define a binary score function $\text{score}(\cdot)$ to judge the usefulness of r for resolving q, where $\text{score}(U_k) = 0$ or 1 means that r is unhelpful or helpful, respectively. In practice, $\text{score}(\cdot)$ is implemented as a LongFormer (Beltagy et al., 2020) based classifier.

Then, for each candidate tool t_i, we fetch all tool

[2]For simplicity, we also call c_i as "the cost of t_i" in the remaining part of this paper.

usages U_k related to t_i in the past experience $\mathcal{E}$ to construct a set of tool usages $\mathcal{E}_{t_i}$. We calculate the weighted average score of all $U_k \in \mathcal{E}_{t_i}$ as the expected value of t_i for the user query q_u:

$$v(q_u, t_i) = \frac{\sum\limits_{U_k \in \mathcal{E}_{t_i}} \exp(\mathrm{sim}(q_u, q))\mathrm{score}(U_k)}{\sum\limits_{U_k \in \mathcal{E}_{t_i}} \exp(\mathrm{sim}(q_u, q))}, \tag{3}$$

where $\mathrm{sim}(q_u, q)$ is the similarity between q_u and q, which can be calculated using the retrieval model.

We also notice that repeatedly using the same tool too many times may be suboptimal within the budget constraint, since we expect that more diverse information can be obtained if different tools are used. Thus, we set an estimated frequency constraint $\tilde{F}(q_u, t_i)$ for each candidate tool t_i and expect that t_i should be used no more than $\tilde{F}(q_u, t_i)$ times. Specifically, if a tool frequently returns useless or erroneous messages, we should prevent the LLM from using it to reduce the cost of tool learning. Thus, we set a threshold τ and set $\tilde{F}(q_u, t_i) = 0$ if $v(q_u, t_i) < \tau$ to filter out the useless tools based on the past experience. Otherwise, for each query q in the past experience where t_i is used in the process of resolving q, we count the number of times $F(q, t_i)$ for which t_i is used, and then calculate the weighted average of $F(q, t_i)$ as the estimated frequency constraint $\tilde{F}(q_u, t_i)$:

$$\tilde{F}(q_u, t_i) = \frac{\sum\limits_{q} \exp(\mathrm{sim}(q_u, q))F(q, t_i)}{\sum\limits_{q} \exp(\mathrm{sim}(q_u, q))}. \tag{4}$$

Finally, we regard the planning process of budget-constrained tool learning as a knapsack problem and use dynamic programming to find a preferable plan of tool usage within the budget constraint. Formally, for each candidate tool t_i, the plan specifies its corresponding frequency $f(q_u, t_i)$, which means that we expect the LLM to use t_i for at most $f(q_u, t_i)$ times during the tool learning process. Since we expect that the returned results should contain valuable information as much as possible, the plan is determined by maximizing the sum of the expected value:

$$V = \sum_{i=1}^{n} f(q_u, t_i)v(q_u, t_i) \tag{5}$$

Algorithm 1 Tool Usage Planning

```
Input: budget constraint $B$; estimated cost for system prompt and user prompt $c_s$; estimated cost $c_i$, expected value $v(q_u, t_i)$ and frequency constraint $\tilde{F}(q_u, t_i)$ for each tool in $\mathcal{T} = \{t_1, \dots, t_i, \dots, t_n\}$
Output: frequency $f(q_u, t_i)$
 1: $R \leftarrow B - c_s$
 2: for $j \leftarrow 0$ to $R$ do
 3:     $V_{0,j} \leftarrow 0$
 4: for $i \leftarrow 1$ to $n$ do
 5:     for $j \leftarrow 0$ to $R$ do
 6:         $V_{i,j} \leftarrow V_{i-1,j}$
 7:         for $k \leftarrow 1$ to $\lfloor \tilde{F}(q_u, t_i) \rfloor$ do
 8:             if $j \geq kc(t_i)$ then
 9:                 $V_{i,j} \leftarrow \max(V_{i,j}, V_{i,j-kc(t_i)} + kv(q_u, t_i))$
10: $V_{max} \leftarrow 0$
11: for $j \leftarrow 0$ to $R$ do
12:     $V_{max} \leftarrow \max(V_{max}, V_{n,j})$
13: Trace back the dynamic programming process to obtain the frequency $f(q_u, t_i)$
```

under the budget and frequency constraints:

$$\sum_{i=1}^{n} f(q_u, t_i)c_i \leq R, \tag{6}$$

$$f(q_u, t_i) \leq \tilde{F}(q_u, t_i), \tag{7}$$

where $R = B - c_s$. For simplicity, we assume that c_i and R are positive integers.[3] The details are shown in Algorithm 1.

The process of finding the plan also introduces extra cost. Fortunately, the cost is a near-constant value. Thus, we can simply extend c_s to include this extra cost and keep the algorithm untouched.

Moreover, it is worth noting that our proposed dynamic programming method can be regarded as a searching algorithm. Searching-based methods are widely used in previously proposed tool learning methods such as DFSDT (Qin et al., 2023b) and ToT-DFS (Yao et al., 2023a). However, these methods are not feasible for searching tool usage plans as they need to invoke the tools during searching, which consume the budget.

[3]If c_i and R are real numbers, we can use their approximate values and change them into integers. Theoretically, this approximation can achieve any precision. For the detailed proof, please refer to Appendix A.1.

Algorithm 2 Budget-Constrained Tool Learning with Plan

Input: user query q_u, set of candidate tools $\mathcal{T} = \{t_1, \ldots, t_i, \ldots, t_n\}$, frequency $f(q_u, t_i)$ for each tool t_i

Output: returned result y

```
1: for each $t_i \in \mathcal{T}$ do
2:     if $f(q_u, t_i) = 0$ then
3:         $\mathcal{T} \leftarrow \mathcal{T} \setminus \{t_i\}$
4: $j \leftarrow 0$
5: while True do
6:     $j \leftarrow j + 1$
7:     $t_{a_j}, p_j \leftarrow$ UseTool($q_u, \mathcal{T}, r_1, \ldots, r_{j-1}$)
           ▷ Select a tool and determine its parameter
8:     $r_j \leftarrow t_{a_j}(p_j)$ ▷ Invoke the tool to get result
9:     $f(q_u, t_{a_j}) \leftarrow f(q_u, t_{a_j}) - 1$
10:    if $f(q_u, t_{a_j}) = 0$ then
11:        $\mathcal{T} \leftarrow \mathcal{T} \setminus \{t_{a_j}\}$
12:    if $\mathcal{T} = \emptyset$ or IsSufficient($q_u, r_1, \ldots, r_j$) then ▷ If there are no available tools in $\mathcal{T}$ or the returned results are sufficient for resolving the user query
13:        $y \leftarrow$ Summarize($q_u, r_1, \ldots, r_j$)
               ▷ Summarize to get the returned result
14:        return $y$
```

2.3 Applying Plan to Tool Learning

After we obtain the frequency $f(q_u, t_i)$ for each candidate tool t_i, we apply the plan to the tool learning process. Specifically, before the LLM uses tools, we remove all the tools t_i which satisfy $f(q_u, t_i) = 0$ from $\mathcal{T}$ since they should not be used in the process. Then, after each step j when the LLM uses the tool t_{a_j}, we reduce the frequency $f(q_u, t_{a_j})$ by 1. Once this frequency reaches zero, we remove t_{a_j} from $\mathcal{T}$, which indicates that t_{a_j} should be no more used. We add extra information in the input context to inform the LLM that the tool t_{a_j} is forbidden to use. If the LLM occasionally uses the forbidden tools, we prevent the LLM from the actual action of the tool usage and return an error message. Finally, if $\mathcal{T}$ becomes empty or the returned results are sufficient to draw a conclusion, we make a summary about all the returned results. The aforementioned process is shown in Algorithm 2.

2.4 Blacklist Mechanism

To further reduce the cost of tool learning, we introduce a blacklist mechanism to reduce the cost of repeatedly using unhelpful tools. During the tool learning process, we temporarily build the blacklist according to the judgment about the returned results. Specifically, after the LLM uses a tool and receives the returned result, we use the classifier introduced in Section 2.2 to judge whether the returned result is helpful in resolving the user query.[4] If a tool returns unhelpful results, we list it into the blacklist and forbid the LLM from using it. Specifically, we inform the LLM that the blacklisted tools are forbidden to use by giving the blacklisted tools in the input context during the tool learning process. If the LLM occasionally uses a blacklisted tool, we prevent the LLM from the actual action of the tool usage and return an error message.

3 Experiments

3.1 Setup

Data Preparation. Our experiments are mainly conducted on the ToolBench (Qin et al., 2023b) dataset, which is a tool learning dataset in English. We follow Qin et al. (2023b) and use 6 different subsets (I1-Inst, I1-Tool, I1-Cat, I2-Inst, I2-Cat and I3-Inst) in ToolBench as the test datasets. The subset I3-Inst includes 100 user queries, while each of the other subsets contains 200 user queries.

For the past experience used for planning the tool usage, we construct a dataset using the training dataset of ToolBench. First, we use ChatGPT (OpenAI, 2022) to judge whether the returned result is helpful for approximately 30k instances of tool usages in the training dataset. Then, we use these instances to train the LongFormer-based classifier proposed in Section 2.2 and use the classifier to classify the remaining instances in the training dataset. The total number of these two parts of instances is about 2.5M. During our experiments, all the past experience is obtained from this dataset.

Baselines. We combine our proposed method with the following baseline methods and compare the combined methods with the baseline methods without such combination:

1. **ReAct** (Yao et al., 2023b): Tool learning is performed using a chain-of-thought process (Wei et al., 2022).

2. **DFSDT** (Qin et al., 2023b): Tool learning is performed using a tree-based depth-first

[4]The cost of the classifier can also be added to c_i.

Method	I1-Inst		I1-Tool		I1-Cat		I2-Inst		I2-Cat		I3-Inst		Average	
	PBC↑	AC↓	PBC↑	AC↓	PBC↑	AC↓	PBC↑	AC↓	PBC↑	AC↓	PBC↑	AC↓	PBC↑	AC↓
ReAct	38.5	15.8	37.0	16.3	38.5	15.0	33.5	15.8	35.0	17.0	22.0	12.6	34.1	15.4
+BTP (*Ours*)	**43.5**	**7.6**	**48.0**	**7.8**	**49.5**	**7.2**	**41.5**	**7.0**	**49.5**	**7.1**	**30.0**	**4.7**	**43.7**	**6.9**
ReAct+Prompt	35.5	16.7	39.5	17.1	43.0	14.4	37.5	14.9	39.5	16.3	24.0	14.9	36.5	15.7
+BTP (*Ours*)	**47.0**	**8.7**	**44.0**	**8.3**	**50.5**	**7.8**	**47.0**	**7.7**	**54.0**	**8.0**	**32.0**	**5.2**	**45.8**	**7.6**
DFSDT	34.0	77.5	29.5	75.1	34.5	70.1	28.0	81.6	28.5	79.4	18.0	85.8	28.8	78.3
+BTP (*Ours*)	**58.0**	**9.6**	**58.0**	**10.3**	**63.5**	**9.5**	**77.0**	**9.5**	**66.5**	**9.8**	**64.0**	**6.4**	**64.5**	**9.2**
DFSDT+Prompt	31.0	69.5	29.5	84.8	36.5	61.2	28.5	68.4	31.0	74.4	17.0	81.3	29.0	73.3
+BTP (*Ours*)	**55.0**	**10.0**	**58.0**	**10.1**	**63.0**	**9.2**	**77.5**	**9.5**	**65.5**	**10.0**	**64.0**	**6.1**	**63.8**	**9.2**
ToT-DFS	11.5	52.4	10.5	52.4	11.0	48.8	7.0	52.4	11.0	53.3	10.0	48.9	10.2	51.4
+BTP (*Ours*)	**55.5**	**11.7**	**59.0**	**12.0**	**60.0**	**11.6**	**76.0**	**11.3**	**70.0**	**11.6**	**64.0**	**6.7**	**64.1**	**10.8**
ToT-DFS+Prompt	10.0	48.2	10.5	50.8	14.5	45.9	6.0	49.0	10.0	51.2	8.0	53.3	9.8	49.7
+BTP (*Ours*)	**54.5**	**11.9**	**59.5**	**12.1**	**56.5**	**11.5**	**74.5**	**11.8**	**64.0**	**11.7**	**64.0**	**6.7**	**62.2**	**11.0**

Table 2: Comparison of **Pass rate under Budget Constraint (PBC)** and **Average Cost (AC)** between the baseline methods and our proposed BTP on the ToolBench (Qin et al., 2023b) dataset. The experiments are conducted **with the budget constraint $R = 20$**.

searching algorithm. This allows the LLM to traverse back when encountering a failure.

3. **ToT-DFS** (Yao et al., 2023a): Tool learning is also performed using a depth-first searching algorithm. Different from DFSDT, ToT-DFS makes a vote across the candidate states and expands the best state at each step.

4. **ReAct+Prompt**, **DFSDT+Prompt** and **ToT-DFS+Prompt**: The cost of the candidate tools and the remaining budget are added to the system prompt of the corresponding method.

Evaluation. We mainly use the following evaluation metrics to evaluate the effect of budget constraint on different methods:

1. **Pass rate under Budget Constraint (PBC)**: The percentage of user queries which can be resolved by the LLM under the given budget constraint. When the solution given by the LLM exceeds the budget constraint, we count it as a **failure**.

2. **Average Cost (AC)**: The average cost used for all user queries in the test dataset.

To better compare the quality of the generated solutions of our method with the baselines, we also evaluate the **Pass Rate (PR)** (Qin et al., 2023b), which measures the percentage of successfully resolved user queries, ignoring the budget constraint. Moreover, we also report the **Rate of Failure due to Budget Constraint (RFBC)** to measure the percentage of user queries which the LLM fails to resolve because of the budget constraint.

Implementation Details. We use ChatGPT (OpenAI, 2022) as the backbone model for tool learning. For each user query, we retrieve 5 candidate tools using the semantic retrieval model provided by Qin et al. (2023b). Since the ToolBench dataset does not explicitly provide the cost of the tools, we randomly set an integer in $[1, 10]$ as the cost c_i for each tool in the tool library. For the budget constraint, we set the value of R to 20 unless explicitly specified. We set the threshold $\tau = 0.15$ to filter out the useless tools. For further implementation details, please refer to Appendix A.2.

One might suggest that for a more accurate evaluation, a real system capable of calculating costs on the fly should be implemented. However, creating such a system involves navigating numerous real-world challenges, including the unpredictability associated with LLM behavior and network dynamics. Given that this work represents an initial exploration into budget-constrained tool learning, postponing the development of this system to future research seems reasonable. Additionally, it is crucial to clarify that the cost values c_i used in the test set above are considered to be aggregate values encompassing all relevant costs, such as monetary expenses, time, and LLM usage, etc. The rationale behind this assumption and its validity is thoroughly discussed at the end of Section 2.1.

3.2 Main Results

The experimental results are shown in Table 2 and 3. When combining our proposed BTP with the baseline methods, the LLM can better resolve the user query within the budget constraint, regardless of which baseline method we use. Specifically, when combining BTP with the baseline methods, the average value of Pass rate under Budget Constraint (PBC) over six subsets of the ToolBench dataset increases by 9.3-53.9 points, and the Average Cost (AC) also decreases by 8.1-69.1 points. This is mainly attributed to two major aspects: First, combining BTP with the baseline methods does not significantly harm the Pass Rate (PR) even if we ignore the budget constraint. Second, the tool learning process cannot be terminated within the budget constraint for a substantial part of the user queries if we do not combine BTP with the baseline methods, while utilizing BTP can guarantee that the tool learning process can be terminated within the budget constraint.

Moreover, adding extra information to the system prompt to remind the LLM about the budget constraint is not effective for budget-constrained tool learning. For example, DFSDT+Prompt only outperforms DFSDT by 0.2 points on PBC. The average cost of DFSDT+Prompt is 73.3, which is much higher than the budget constraint $R = 20$.

Method	**PR↑**	**RFBC↓**
ReAct	**44.0**	26.4
+BTP (*Ours*)	43.7	**0.0**
ReAct+Prompt	44.5	23.9
+BTP (*Ours*)	**45.8**	**0.0**
DFSDT	63.8	60.3
+BTP (*Ours*)	**64.5**	**0.0**
DFSDT+Prompt	**63.8**	59.2
+BTP (*Ours*)	**63.8**	**0.0**
ToT-DFS	61.6	86.3
+BTP (*Ours*)	**64.1**	**0.0**
ToT-DFS+Prompt	**63.1**	86.7
+BTP (*Ours*)	62.2	**0.0**

Table 3: Comparison of **Pass Rate (PR)** and **Rate of Failure due to Budget Constraint (RFBC)** between the baseline methods and our proposed BTP. The experiments are conducted **with the budget constraint $R = 20$**. The detailed results are shown in Appendix A.3.

Method	**AC**	**#Use**
Plan	13.9	2.9
ReAct+BTP	7.6	1.5
ReAct+Prompt+BTP	8.7	1.7
DFSDT+BTP	9.6	1.8
DFSDT+Prompt+BTP	10.0	1.9
ToT-DFS+BTP	11.7	2.3
ToT-DFS+Prompt+BTP	11.9	2.4

Table 4: Comparison of the planned and the actual tool usage on the I1-Inst subset. "#Use" represents the average number of times the LLM uses the tools for each user query.

3.3 Planned v.s. Actual Tool Usage

In this section, we compare the plan given by the dynamic programming algorithm and the actual tool usage performed by the LLM. As shown in Table 4, the average cost of the plan is 13.9 and we expect that the LLM uses the tools for 2.9 times on average on the I1-Inst subset. When BTP is combined with the baseline methods, both the actual average cost and the average number of times the LLM uses the tools are less than those given in the plan. The gap between the planned and the actual tool usage is within our expectation, since we do not actually use the tools when making the plan. Moreover, the size of the gap is reasonable, especially when we use a searching based algorithm (DFSDT or ToT-DFS) for tool learning.

3.4 Effect of Budget Constraint R

To investigate how the budget constraint R affects the performance of budget-constrained tool learning, we also conduct experiments on the I1-Inst subset under different budget constraints. The results are shown in Figure 2. On the one hand, the value of PBC increases as the budget constraint does, since a higher budget constraint gives more chances to the LLM for using the tools to resolve the user query. On the other hand, the average cost will reduce if we set a lower budget constraint, which indicates that we can adjust the cost of tool learning by modifying the budget constraint.

3.5 Effect of Threshold τ

We introduce a threshold τ in Section 2.2 to filter out useless tools. The effect of τ on the I1-Inst subset is shown in Figure 3. First, if we set a higher threshold, the average cost decreases since fewer tools are available for the LLM to use. Second, the value of PBC increases when $\tau < 0.15$, which indicates that setting a proper threshold can filter out the useless tools and improve the quality of the generated solutions. However, the value of PBC

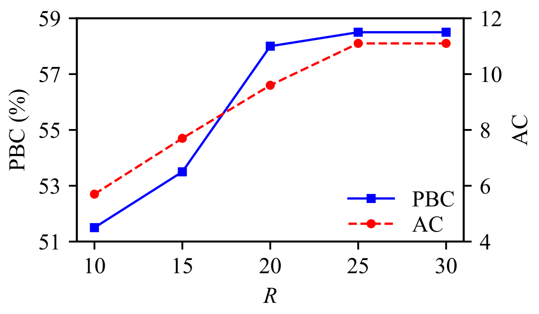


Figure 2: Effect of budget constraint R on Pass rate under Budget Constraint (PBC) and Average Cost (AC). The results are evaluated on the I1-Inst subset.

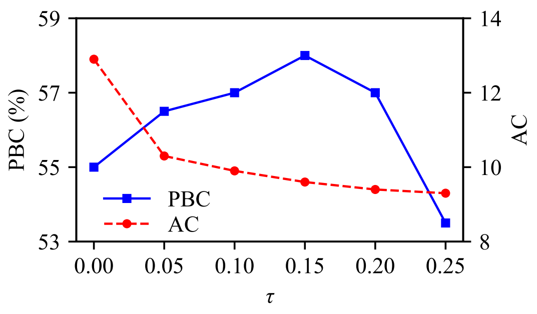


Figure 3: Effect of threshold τ on Pass rate under Budget Constraint (PBC) and Average Cost (AC). The results are evaluated on the I1-Inst subset.

decreases when $\tau > 0.15$, which indicates that a high threshold may incorrectly filter out some useful tools and worsen the quality of the solutions. Therefore, we set $\tau = 0.15$ in our experiments.

3.6 Effect of Blacklist Mechanism

To validate the effectivenss of the blacklist mechanism, we conduct an extra experiment on the I1-Inst subset where the blacklist mechanism is removed. As shown in Table 5, removing the blacklist does not significantly affect the value of PBC. However, the average cost increases, which indicates that the blacklist mechanism can reduce the cost of repeatedly using unhelpful tools.

Moreover, to investigate how the LongFormer-based classifier affects the performance of budget-constrained tool learning, we conduct an extra experiment on the I1-Inst subset where we directly use the LLM to judge whether the returned result is helpful in the blacklist mechanism. Experimental results show that the value of PBC and the average cost do not significantly change if we use the LLM to judge the helpfulness of the returned results, even if the LLM can give the judgments more accurately than the classifier.[5] However, using the LongFormer-based classifier can reduce the number of times of using the LLM in practice, and thus we use the LongFormer-based classifier for our proposed BTP.

Blacklist	**PBC**↑	**AC**↓
None	57.5	10.2
ChatGPT	**58.0**	9.7
LongFormer	**58.0**	**9.6**

Table 5: Effect of blacklist mechanism on I1-Inst.

4 Related Work

Tool learning aims to extend the capability of LLMs using external tools (Schick et al., 2023; Yao et al., 2023b; Qin et al., 2023a,b). With the aid of external tools, LLMs may be capable of obtaining recent information (Liu et al., 2023), solving difficult math problems (Cobbe et al., 2021; Gao et al., 2023b), handling multimodal information (Huang et al., 2023; Lu et al., 2023) or solving domain-specific problems (Jin et al., 2023).

The basic approach of tool learning is to add the documentation of the tool to the input context of the LLM and then let the LLM use the tool to resolve the user query (Shen et al., 2023; Hsieh et al., 2023; Qin et al., 2023b). To handle complex user queries which require multiple steps to resolve, Yao et al. (2023b) propose ReAct, which performs tool learning in a chain-of-thought (Wei et al., 2022) process. To increase the probability of successfully resolving the user query, Qin et al. (2023b) propose DFSDT, a tool learning algorithm which is based on a tree-based depth-first searching algorithm and allows the LLM to traverse back after a failure. Similarly, Zhuang et al. (2023) also propose a tree-based algorithm for tool learning, which is based on the A* searching algorithm. Moreover, Gao et al. (2023a) and Gao et al. (2023c) add a reflection mechanism to tool learning, which allows the LLM to retry after a failure.

However, the aforementioned studies focus on how to better resolve complex user queries, but do not take the budget constraint into consideration. In contrast, our proposed method aims to improve the performance of tool learning within the budget constraint of tool usage.

This work is also highly related to Kim et al. (2023), who use different tools in parallel to im-

[5]We randomly sample 100 returned results and use the classifier and the LLM to give the judgments, and we find the accuracy of the classifier and the LLM is 91% and 94%, respectively.

prove the efficiency of tool learning. However, their work mainly focuses on time efficiency rather than budget constraint.

5 Conclusion

In this work, we propose BTP, a novel method for budget-constrained tool learning. The core idea of BTP is to find a preferable plan under the budget constraint before using the tools. We also introduce a blacklist mechanism to further reduce the cost of tool learning. Experimental results show that BTP can be combined with different tool learning methods and improve the performance of budget-constrained tool learning.

Limitations

In this work, we propose a simplified yet effective approximation to model budget-constrained tool learning. As discussed in Section 2.1, our approach does not account for costs incurred by external factors, which are challenging to estimate based on previous experience of tool learning. Exploring a more comprehensive mathematical model that includes these costs is an avenue for future research. Furthermore, our experiments are carried out in a simulated setting, where the costs associated with the tools are not based on real-world data but are simulated instead. Creating similar conditions in a real-world context to accurately estimate tool costs poses numerous challenges. Therefore, developing techniques to estimate real-world tool costs is an objective for future research.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61925601, 62276152). We thank Zonghan Yang for providing helpful suggestions about the idea of this work. We also appreciate the anonymous reviewers for their valuable comments.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

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

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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. 2023a. AssistGPT: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. PAL: Program-aided language models. In *Proc. of ICML*, volume 202, pages 10764–10799.

Zhi Gao, Yuntao Du, Xintong Zhang, Xiaojian Ma, Wenjuan Han, Song-Chun Zhu, and Qing Li. 2023c. CLOVA: A closed-loop visual assistant with tool usage and update. *arXiv preprint arXiv:2312.10908*.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

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

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

Rongjie Huang, Mingze Li, Dongchao Yang, Jiatong Shi, Xuankai Chang, Zhenhui Ye, Yuning Wu, Zhiqing Hong, Jiawei Huang, Jinglin Liu, Yi Ren, Zhou Zhao, and Shinji Watanabe. 2023. AudioGPT: Understanding and generating speech, music, sound, and talking head. *arXiv preprint arXiv:2304.12995*.

Qiao Jin, Yifan Yang, Qingyu Chen, and Zhiyong Lu. 2023. GeneGPT: Augmenting large language models with domain tools for improved access to biomedical information. *arXiv preprint arXiv:2304.09667*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An LLM compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. WebGLM: Towards an efficient web-enhanced question answering system with human preferences. In *Proc. of SIGKDD*, pages 4549–4560.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jian-feng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

OpenAI. 2022. OpenAI: Introducing ChatGPT.

Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hanna Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. ART: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

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, Shi Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bo Li, Ziwei Tang, Jing Yi, Yu Zhu, Zhen-ning Dai, Lan Yan, Xin Cong, Ya-Ting Lu, Weilin Zhao, Yuxiang Huang, Jun-Han Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tong-shuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789v1*.

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugging-GPT: Solving AI tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proc. of NeurIPS*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *Proc. of ICLR*.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. ToolChain*: Efficient action space navigation in large language models with a* search. *arXiv preprint arXiv:2310.13227*.

A Appendix

A.1 Proof of Approximation of c_i and R

In this proof, we suppose that the tool learning process consists of a reasonable number of steps, and we use N to represent the number of steps. For any $\varepsilon > 0$, we set λ as:

$$\lambda = \frac{N+1}{\varepsilon} \tag{8}$$

Then, we use $\tilde{c}_i$ and $\tilde{R}$ to denote the approximate value of c_i and R. $\tilde{c}_i$ and $\tilde{R}$ are defined as below:

$$\tilde{c}_i = \frac{\lceil \lambda c_i \rceil}{\lambda}, \tag{9}$$

$$\tilde{R} = \frac{\lfloor \lambda R \rfloor}{\lambda}. \tag{10}$$

Based on the equations above, $\tilde{c}_i$ and $\tilde{R}$ satisfy $\lambda \tilde{c}_i \in \mathbb{Z}$ and $\lambda \tilde{R} \in \mathbb{Z}$.

Besides, $\tilde{c}_i$ and $\tilde{R}$ satisfy

$$\begin{aligned} R - \tilde{R} &= \frac{\lambda R - \lfloor \lambda R \rfloor}{\lambda} \\ &< \frac{1}{\lambda} \\ &= \frac{\varepsilon}{N+1}, \end{aligned} \tag{11}$$

$$\begin{aligned} \tilde{c}_i - c_i &= \frac{\lceil \lambda c_i \rceil - \lambda c_i}{\lambda} \\ &< \frac{1}{\lambda} \\ &= \frac{\varepsilon}{N+1}. \end{aligned} \tag{12}$$

Thus, the approximate remaining budget $\tilde{R} - \sum_{j=1}^{N} \tilde{c}_{a_j}$ satisfies

$$\begin{aligned} &\tilde{R} - \sum_{j=1}^{N} \tilde{c}_{a_j} \\ &> R - \frac{\varepsilon}{N+1} - \sum_{j=1}^{N} \left(c_{a_j} + \frac{\varepsilon}{N+1} \right) \\ &= R - \sum_{j=1}^{N} c_{a_j} - \varepsilon. \end{aligned} \tag{13}$$

Moreover, since $\tilde{R} = \frac{\lfloor \lambda R \rfloor}{\lambda} \leq R$ and $\tilde{c}_i = \frac{\lceil \lambda c_i \rceil}{\lambda} \geq c_i$, $\tilde{R} - \sum_{j=1}^{N} \tilde{c}_{a_j}$ also satisfies

$$\tilde{R} - \sum_{j=1}^{N} \tilde{c}_{a_j} \leq R - \sum_{j=1}^{N} c_{a_j}. \tag{14}$$

Thus, we conclude that

$$\left| \left(\tilde{R} - \sum_{j=1}^{N} \tilde{c}_{a_j} \right) - \left(R - \sum_{j=1}^{N} c_{a_j} \right) \right| < \varepsilon. \tag{15}$$

This indicates that if c_i and R are real numbers, we can use $\lambda \tilde{c}_i$ and $\lambda \tilde{R}$ as their integral approximation. If the number of the total steps taken for tool learning N is reasonable, such approximation can achieve sufficiently high accuracy if ε is small enough.

A.2 Further Implementation Details

The training process of the LongFormer-based classification model is implemented on top of the Transformers (Wolf et al., 2020) library. The model is initialized with `longformer-base-4096`[6], which has approximately 149M parameters. The model is trained for 10 epochs with the learning rate of 10^{-5}. The batch size is set to 8. The model is trained on a single NVIDIA GeForce RTX 3090 GPU for 5.5 hours.

Moreover, the system prompt used during in the tool learning process is constructed based on the prompt for solution path annotation introduced in Qin et al. (2023b). We add extra content presented in Figure 4 in the system prompt if some of the tools are forbidden to use. Specifically, we append the prompt **Part I** to the system prompt if at least one candidate tool is removed from $\mathcal{T}$. We append the prompt **Part II** to the system prompt if at least one candidate tool is blacklisted. We append the prompt **Part III** to the system prompt if all the candidate tools are forbidden to use (i.e. removed from $\mathcal{T}$ or blacklisted).

A.3 Detailed Experimental Results of Table 3

In this section, we report the detailed experimental results of Table 3 in Table 6 and 7.

A.4 Case Study

In this section, we conduct a case study on the I1-Inst subset. As shown in Figure 5, the baseline method DFSDT repeatedly uses the tool 2 which always returns error messages. DFSDT also uses the tools 3 and 4 which return repetitive or irrelevant results. As the result, DFSDT finishes the tool learning process with the total cost of 31. It only finds the list of supported regions but fails to find the trending keywords in the United Kingdom.

[6]https://huggingface.co/allenai/longformer-base-4096

When combining our proposed BTP with DFSDT, the LLM uses the tool 5 within the plan and successfully finds the trending keywords in the United Kingdom. Thus, DFSDT+BTP successfully resolves the user query with the total cost of 8. This indicates that our proposed BTP can help the LLM resolve the user query within the budget constraint.

A.5 Licenses of Tools, Models and Datasets

The licenses of the Transformers (Wolf et al., 2020) library, the LongFormer (Beltagy et al., 2020) model and the ToolBench (Qin et al., 2023b) dataset are Apache-2.0.

Part I

```
Now some subfunctions of the available tools are forbidden since using them will
exceed the budget limitation. The forbidden subfunctions are listed as below:
<List of tools removed from 𝒯>
```

Part II

```
Now some subfunctions of the available tools are forbidden since they cannot
provide valuable information or usually return error messages. The forbidden
subfunctions are listed as below:
<List of blacklisted tools>
```

Part III

```
Now all subfunctions of the available tools are forbidden, and you should call
the "Finish" function to end the task.
```

Figure 4: The extra content which may be appended to the system prompt during the tool learning process.

Method	I1-Inst	I1-Tool	I1-Cat	I2-Inst	I2-Cat	I3-Inst	Average
ReAct	47.0	51.5	45.0	45.0	47.5	28.0	44.0
+BTP (*Ours*)	43.5	48.0	49.5	41.5	49.5	30.0	43.7
ReAct+Prompt	43.5	52.5	48.0	44.5	49.5	29.0	44.5
+BTP (*Ours*)	47.0	44.0	50.5	47.0	54.0	32.0	45.8
DFSDT	56.0	62.0	56.5	76.0	68.5	64.0	63.8
+BTP (*Ours*)	58.0	58.0	63.5	77.0	66.5	64.0	64.5
DFSDT+Prompt	54.0	63.5	60.0	76.0	65.5	64.0	63.8
+BTP (*Ours*)	55.0	58.0	63.0	77.5	65.5	64.0	63.8
ToT-DFS	53.5	60.0	54.5	73.5	65.0	63.0	61.6
+BTP (*Ours*)	55.5	59.0	60.0	76.0	70.0	64.0	64.1
ToT-DFS+Prompt	54.5	61.5	58.5	74.5	66.5	63.0	63.1
+BTP (*Ours*)	54.5	59.5	56.5	74.5	64.0	64.0	62.2

Table 6: Comparison of **Pass Rate (PR)** between the baseline methods and our proposed BTP on the ToolBench (Qin et al., 2023b) dataset. The experiments are conducted **with the budget constraint $R = 20$**.

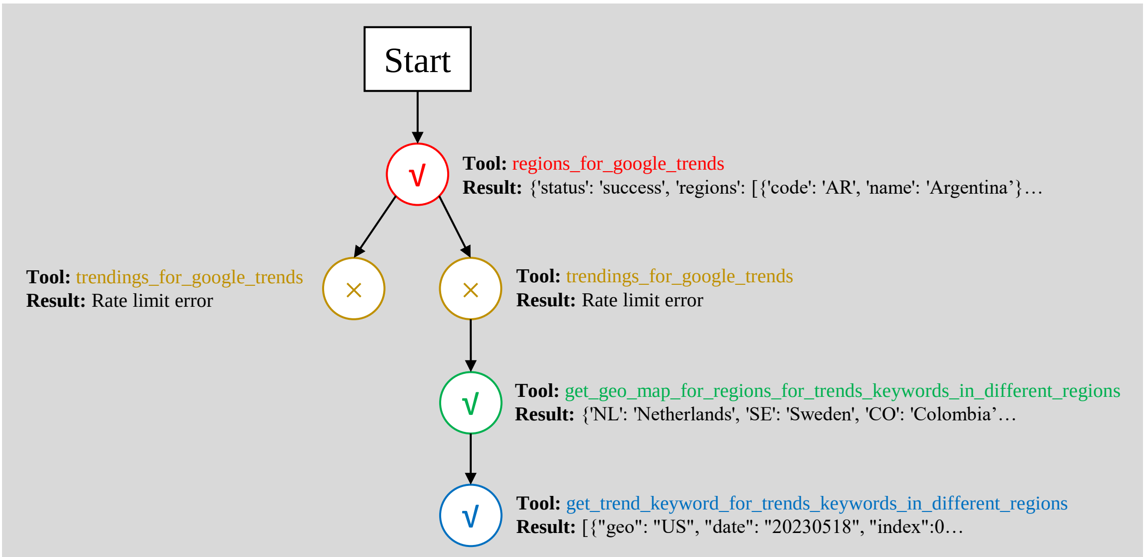
Method	I1-Inst	I1-Tool	I1-Cat	I2-Inst	I2-Cat	I3-Inst	Average
ReAct	27.0	30.0	22.5	28.5	31.5	19.0	26.4
ReAct+Prompt	27.0	30.0	17.5	20.5	24.5	24.0	23.9
DFSDT	56.0	61.0	53.5	62.0	63.5	66.0	60.3
DFSDT+Prompt	54.5	62.5	53.0	57.0	57.0	71.0	59.2
ToT-DFS	85.5	84.5	87.0	92.5	87.0	81.0	86.3
ToT-DFS+Prompt	85.0	86.5	81.0	91.0	86.5	90.0	86.7

Table 7: **Rate of Failure due to Budget Constraint (RFBC)** of the baseline methods on the ToolBench (Qin et al., 2023b) dataset. The experiments are conducted **with the budget constraint $R = 20$**. For our proposed BTP, the value of RFBC is consistently equal to 0 since the plan made in Algorithm 1 guarantees that the tool learning process can be finished within the budget constraint.

User Query:
I'm curious about the trending search keywords on Google for today in the United Kingdom. Can you please fetch the list of trending keywords? Additionally, I would like to know if a specific region is supported. Can you provide me with the list of supported regions?

Candidate Tools (Cost):
1. regions_for_google_trends (2)
2. trendings_for_google_trends (9)
3. get_geo_map_for_regions_for_trends_keywords_in_different_regions (3)
4. get_trend_keyword_for_trends_keywords_in_different_regions (8)
5. keywordsearch_for_google_keyword_scraper (6)

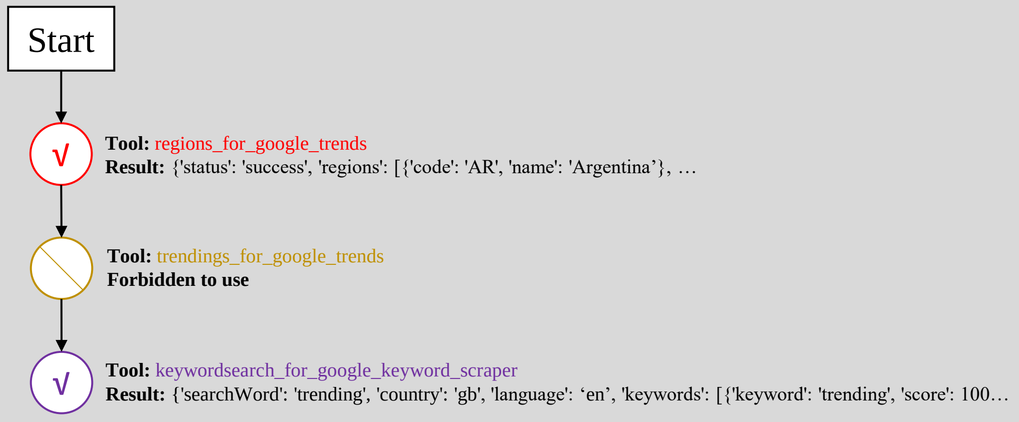
Execution result of DFSDT (Total cost: 31):


Start
√
Tool: regions_for_google_trends
Result: {'status': 'success', 'regions': [{'code': 'AR', 'name': 'Argentina'}…
×
Tool: trendings_for_google_trends
Result: Rate limit error
×
Tool: trendings_for_google_trends
Result: Rate limit error
√
Tool: get_geo_map_for_regions_for_trends_keywords_in_different_regions
Result: {'NL': 'Netherlands', 'SE': 'Sweden', 'CO': 'Colombia'…
√
Tool: get_trend_keyword_for_trends_keywords_in_different_regions
Result: [{"geo": "US", "date": "20230518", "index":0…

Summary:
The list of supported regions for trending keywords includes Argentina, Australia … and Vietnam. Regarding the trending search keywords for the United Kingdom, the rate limit was exceeded for the specified date (2023-09-18 and 2023-09-20), so the trending keywords could not be retrieved at this time.

Execution result of DFSDT+BTP (Total cost: 8):

Plan:
regions_for_google_trends ×1 keywordsearch_for_google_keyword_scraper ×3


Start
√
Tool: regions_for_google_trends
Result: {'status': 'success', 'regions': [{'code': 'AR', 'name': 'Argentina'}, …
Tool: trendings_for_google_trends
Forbidden to use
√
Tool: keywordsearch_for_google_keyword_scraper
Result: {'searchWord': 'trending', 'country': 'gb', 'language': 'en', 'keywords': [{'keyword': 'trending', 'score': 100…

Summary: The trending search keyword for today in the United Kingdom is 'trending', with a score of 100. Other related trending keywords include 'trending 2023', 'trending travel', 'trending gifts', and more. The supported regions for trending search data include Argentina, Australia … and more.

Figure 5: Case study on the I1-Inst subset.