ToolNet: Connecting Large Language Models with Massive Tools via Tool Graph

Anonymous ACL submission

Abstract

While achieving remarkable progress in a broad range of tasks, large language models (LLMs) remain significantly limited in properly using massive external tools. Existing in-context learning approaches simply format tools into a list of plain text descriptions and input them to LLMs, from which, LLMs generate a sequence of tool calls to solve problems step by step. Such a paradigm ignores the intrinsic dependency between tools and offloads all reasoning loads to LLMs, making them restricted to a limited number of specifically designed tools. It thus remains challenging for LLMs to operate on a library of massive tools, casting a great limitation when confronted with real-world scenarios. This paper proposes *ToolNet*, a plug-and-play framework that scales up the number of tools to thousands with a moderate increase in token consumption. *ToolNet* organizes tools into a directed graph. Each node represents a tool, and weighted edges denote tool transition. Starting from an initial tool node, an LLM navigates in the graph by iteratively choosing the next one from its successors until the task is resolved. Extensive experiments show that *ToolNet* can achieve impressive results in challenging multi-hop tool learning datasets and is resilient to tool failures.

1 Introduction

There is an emerging interest (Qin et al., 2023; Song et al., 2023; Yao et al., 2022; Patil et al., 2023; Yang et al., 2023) in unleashing the power of large language models (LLMs) to effectively interact with various tools (or APIs) for real-world tasks. Tool-augmented LLMs, when connected feasibly with massive APIs, can serve as an intermediate interface between average humans and complex tools, which may eventually reshape the vast ecosystem of applications. This endeavor has yielded several impressive industrial outcomes, such as New Bing the search engine (Microsoft Corporation, 2023b), Copilot the office assistant (Microsoft Corporation, 2023a), RT-2 the robot controller (Brohan et al., 2023), and WebGPT the web-browsing agent (Nakano et al.).

Despite the remarkable progress, tool-augmented LLMs are still in the experimental stage and not yet ready to fully meet real-world demands. Notably, while LLMs are designed as generalists for multiple tasks, LLM-powered agents are commonly customized with few-shot in-context examples for narrow purposes. They are limited to connecting with a small number of specially designed tools. For example, Toolformer (Schick et al., 2023) masters 5 tools such as calculator, question-answering engine, calendar, etc. Chameleon (Lu et al., 2023) specializes in two knowledge-intensive question-answering tasks with a set of well-curated tools, such as table verbalizer and image captioner.

Scaling up the number of tools to thousands is challenging for LLMs. This is not merely due to the surging token consumption when tools are prompted as input to LLMs, which commonly exceeds their token limits. More importantly, it is beyond LLMs' capability to select the correct one(s) from a library of vast tools through straightforward in-context learning. According to (Hao et al., 2023), as the number of tools increases, LLMs tend to hallucinate and make mistakes in calling tools, causing steady performance degradation. In response, researchers put huge efforts into training LLMs to master massive tools (Qin et al., 2023; Patil et al., 2023; Hao et al., 2023). While this method could yield promising results, it brings prohibitive computation costs and lacks adaptability to new tools or tools with constant function updates. Moreover, in a vast tool library, low-quality tools should indeed be reckoned with. When misled by tools, LLMs are prone to hallucinate. In response, (Xie et al., 2023) employs beam search to explore the most proper tool(s). However, token consump-

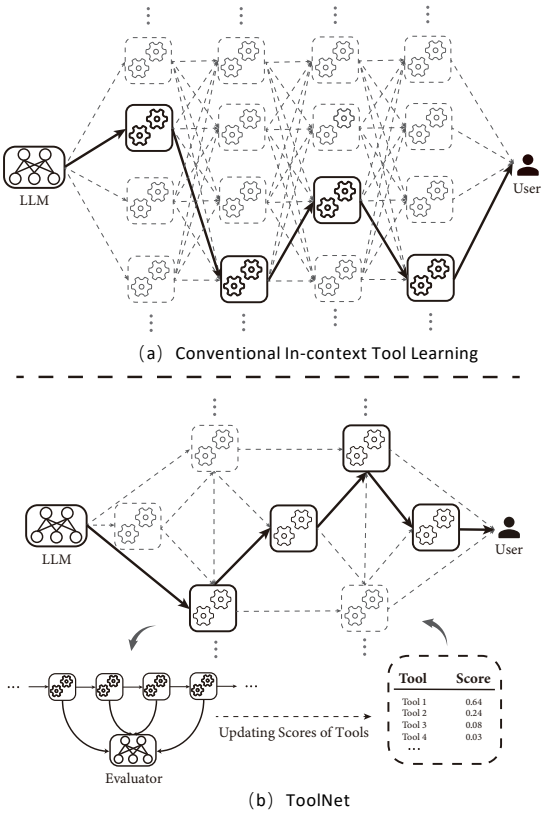


Figure 1: Comparison of (a) prior in-context tool learning methods and (b) the proposed *ToolNet*. Conventionally, all tools are formatted as the input to an LLM for step-by-step tool calling, posing scalability challenges for using massive tools. *ToolNet* organizes tools into a graph. An LLM chooses only from the successor tools relative to its previous selection. An Evaluator assesses the effectiveness of tools and dynamically modifies the tool transition weights within the graph.

tion is multiplied, and without an explicit memory mechanism, the exploration experience cannot be leveraged for subsequent tasks. To address the issues above, we raise the following questions:

Question 1. *How to enable an LLM to cope with massive tools while retaining token efficiency?*

We analyze the multi-hop tool-use trajectories in ToolBench (Qin et al., 2023), recognized as the most extensive publicly available dataset dedicated to tool learning. As depicted in Figure 2, our analysis reveals that tools generally have a restricted set of potential successor tools to be invoked. This observation signifies the sparse transition between tools. In essence, when a particular tool is invoked, the subsequent tool to be called can be constrained to a remarkably limited set of options. The present study leverages this tool-use pattern to connect an LLM with massive tools and retain token efficiency.

Question 2. *How can an LLM identify ineffective tools and modify its tool-utilization strategies for future tasks?*

(Shinn et al., 2023) shows that LLMs can verbally reflect their tool-use trajectories for improvements within ongoing interactions or episodes. In this work, we consider fine-grained reflection at the tool level. Every tool-use step is probed and scored by an LLM. Scores will be used to adjust the tool transition weights, where low-quality tools will be restricted with reduced transition weights. Notably, (Shao et al., 2023) shows that providing an LLM with scores of candidate answers brings significant performance improvements. Likewise, we prompt an LLM with the transition weights of tools. In this way, by encoding the experience into a tool graph, failure calls to low-quality tools can be significantly reduced.

To this end, this paper presents *ToolNet*, a simple yet effective paradigm that assists LLMs in handling massive tools, learning to select appropriate tools, and avoiding calls to broken tools. A comparison between *ToolNet* and other methods is provided in Table 1. As is shown in Figure 1, *ToolNet* organizes the tools in a weighted directed graph. The graph is built based on the tool-use trajectories produced by an LLM. In turn, the LLM uses the graph to reason its tool calls, a process that can be conceptualized as navigating within this graph. Furthermore, the graph can be online updated, thereby enabling its adjustment to accommodate the frequent updates of tools or the introduction of new tasks. Extensive experiments are conducted on five distinct datasets: SciQA (Lu et al., 2022a), TabMWP (Lu et al., 2022b), MATH (Hendrycks et al., 2021), APIBank (Li et al., 2023), and ToolBench (Qin et al., 2023). The results indicate that *ToolNet* consistently outperforms its counterparts in terms of overall performance. Notably, it demonstrates remarkable resilience against the interference of noisy tools and achieves this superior performance while utilizing significantly fewer tokens.

2 Problem Formulation

Tool-augmented LLMs interact with the environment through structured texts in natural language. The general interaction process can be described as follows: at step s, an LLM takes the current environmental observation o_s along with a history of interaction activities $\mathcal{C}_s$ as the input and produces a verbal thought $t_s \in \mathcal{L}$. Then, the LLM interacts with the environment by taking an action

Table 1: A comparison of work that augments LLMs with tool usage. *adapt.* denotes the adaptability to new tools or tools with updates. NL denotes natural language. Plug-Play means the LLMs can be equipped and unequipped with a tool flexibly.

Method	API/Tool Use			Framework	
	#APIs	Adapt.	Token Efficiency	Planning	Plug. Play
Toolformer	5	–	+	-	✗
ReAct	3	+	-	NL	✓
ToT	-	–	–	NL	✓
Reflexion	-	+	–	NL	✓
Chameleon	15	-	-	NL	✓
HuggingGPT	24	+	-	NL	✓
Gorilla	1645	-	+	-	✗
RestGPT	100+	+	-	NL	✓
ToolNet (**ours**)	3992	++	+	NL+Graph	✓

$a_s \in \mathcal{A}$, which can be described as a tuple of a tool name and its arguments, e.g., $a_s = (tool_s, arg_s)$, where $tool_s \in \mathcal{T}$. Subsequently, the LLM gets a new observation o_{s+1} from the environment $\mathcal{E}$. By iterating the interaction process, the LLM may eventually resolve a given task.

LLMs are mostly stateless. The history of interactions $\mathcal{C}_s$ is generally formulated as part of the input to LLMs. It helps an LLM recall the previous process and infer the next action a_{s+1}. We define $\mathcal{C}_s$ as a queue of size K, consisting of observations, thoughts and actions in the past, e.g., $\mathcal{C}_s = [(o_{s-k}, t_{s-k}, a_{s-k}), \cdots, (o_{s-1}, t_{s-1}, a_{s-1})]$. In each step, a tuple of new thought, action, and observation is appended to $\mathcal{C}_s$, and if the maximum capacity is reached, the earliest tuple is popped out. In addition to $\mathcal{C}_s$, the set of available tools $\mathcal{T}$ needs to be included as the input context as well, from which, the LLM may choose a proper one.

Let $\mathcal{P}_\theta$ denotes a tool-augmented LLM with parameters θ. Formally, the interaction process described above can be formulated as follows.

$$t_s = \mathcal{P}_\theta(\mathcal{C}_s \cup o_s), \tag{1}$$

$$a_s = \mathcal{P}_\theta(\mathcal{C}_s \cup o_s, t_s, \mathcal{T}), \tag{2}$$

$$o_{s+1} = \mathcal{E}(a_s) \tag{3}$$

A predominant issue of existing tool-augmented LLMs is their heavy token consumption, which can be largely attributed to the long input context of $\mathcal{C}_s$ and $\mathcal{T}$. A representative solution is by segmenting the long texts of $\mathcal{C}_s$ into paragraphs and compressing them into semantic embeddings. Subsequently, the semantic similarity scores between the candidate paragraphs and the current observation o_s can be computed and ranked. Finally, $\mathcal{C}_s$ in Eq. 1- 2 is replaced by the most relevant paragraphs as the input to LLMs. Intuitively, this solution can be extended to $\mathcal{T}$, e.g., searching tools in $\mathcal{T}$ that are the most relevant to the thought t_s. However, according to Eq. 1, t_s is generated without the full knowledge of $\mathcal{T}$. The semantic information in t_s can be irrelevant to any tools in $\mathcal{T}$. Moreover, in a library of massive API functions, there can be many tools sharing similar semantic information but differing subtly in functionalities. Searching tools by semantic embeddings can be coarse and inaccurate when $|\mathcal{T}|$ goes large. Consequently, existing tool learning approaches either are restricted to a limited number of specifically designed tools or use instruction finetuning to augment the LLM $\mathcal{P}_\theta$ with the prior knowledge of $\mathcal{T}$, which can be computationally intensive.

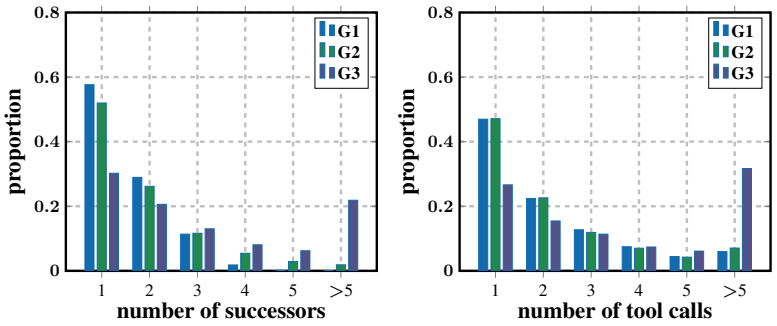
3 ToolNet

The idea of *ToolNet* is simple. Instead of inputting all tools in $\mathcal{T}$ to $\mathcal{P}_\theta$ as in Eq. 2, we provide only a subset $\mathcal{T}_s \subset \mathcal{T}$, according to a policy π and the previous action a_{s-1}, i.e.,

$$a_s = \mathcal{P}_\theta(\mathcal{C}_s \cup o_s, t_s, \mathcal{T}_s), \tag{4}$$

where $\mathcal{T}_s = \pi(\mathcal{T}, a_{s-1})$. Markov assumption takes effect in the generation of $\mathcal{T}_s$. We describe π as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that connects all the tools in $\mathcal{T}$. $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and edges, respectively. Every tool in $\mathcal{T}$ is regarded as a unique node in $\mathcal{V}$. In addition, two special nodes, $start$ and end, are added into $\mathcal{V}$. In other words, $\mathcal{V} = \mathcal{T} \cup \{start, end\}$. An edge $e = (v_i, v_j, w_{i,j})$ connects node v_i to node v_j, with $w_{i,j}$ denotes the transition weight. Note that tools along with the transition weights are formatted as the input texts to LLMs. All nodes except $start$ and end have a self-transition edge that directs to themselves. The $start$ node directs to all the other nodes, which in turn directs to the end.

To this end, an LLM taking actions is the same as traveling in the graph $\mathcal{G}$. Assume the LLM used $tool_{s-1} = v_i$ in the last iteration. The available tools $\mathcal{T}_s$ in the next step are simply the out-neighbor nodes of v_i, e.g., $\mathcal{T}_s = oneigh(v_i)$. If the graph $\mathcal{G}$ is sparse, $|\mathcal{T}_s| \ll |\mathcal{T}|$, bringing token efficiency to tool-augmented LLMs. Notably, the $start$ node connects all tools. In other words, the LLM starts with $\mathcal{T}_1 = \mathcal{T}$, which bottlenecks the token consumption and becomes impractical as $|\mathcal{T}|$ scales up. To handle this problem, we leverage the semantic similarity search approach mentioned in Section 2. The



(a) (b)

Tool	Category	#Successors
jokes/random	Social	49
jokes/search	Social	33
weather/current	Search	31
soundcloud	Search	29
currency exchange	Search	26
Reddit meme	Social	24
Date	Search	22
weather/forecast	Search	22
nutrition analysis	Health	20

(c)

Figure 2: The tool-use statistics on Toolbench, which consists of three subsets: G1, G2, and G3. (a) shows that around 80% of tools have less than 6 successor tools called. (b) illustrates that over 90% of tools in the G1 and G2 subsets, as well as over 50% of tools in the G3 subset, are called fewer than 6 times. (c) lists tools with the most successor tools in the G3 subset. These statistics motivate the proposed *ToolNet* to construct a tool graph with sparse connections of tools.

description of each tool is compressed into a semantic embedding, which is then ranked based on its similarity to the embedding of the initial task description o_1. Only the K most relevant tools are kept to set up $\mathcal{T}_1$.

An additional benefit introduced by *ToolNet* is the transition weights of tools in $\mathcal{T}_s$, which measure the prior preference of tools. Based on the weights, tools can be sorted in order and formatted as the input context. The LLM can refer to the preference scores to make selections. In contrast, the existing approaches, such as ReAct and Reflexion, provide an LLM with tools of equal preference. The LLM relies solely on its internal reasoning ability to make a choice.

3.1 Graph Construction

The construction of the graph $\mathcal{G}$ is vital to the performance and efficiency. If all tools in $\mathcal{G}$ are mutually connected with equal edge weights, *ToolNet* becomes uninformative and degrades into the naive ReAct approach. A good graph $\mathcal{G}$ should be sparse, i.e., nodes have a small number of out-neighbors. Out-of-date tools that are no longer functional should be downweighted. Notably, graph construction can be flexible. This paper considers static and dynamic graph construction approaches. Other graph operations, such as composition, pruning, and partition, are worthy of exploration but beyond the scope of this paper.

3.2 Static Construction

Static construction requires large amounts of tool-use trajectories, such as the massive handwritten code snippets on GitHub that call PyTorch API functions. The orders of API function calls can be leveraged to build up $\mathcal{G}$. Tool-use trajectories can also be generated by tool-augmented LLMs. The multi-step reasoning trajectories represent sequences of tool calls. Several existing works such as ToolBench (Qin et al., 2023), APIBench (Peng et al., 2022), API-Bank (Li et al., 2023), and ToolAlpaca (Tang et al.), have released their curated tool-use instances, from which *ToolNet* can be constructed. Unfortunately, the quality of trajectories generated from LLMs is of great concern.

The static construction of $\mathcal{G}$ is straightforward. It is the same as 2-gram language modeling. Denote $\mathcal{D}$ as a set of tool-use trajectories that have completed their tasks. A trajectory has a sequence of tool uses, e.g., $[tool_1, \cdots, tool_s, \cdots, end]$, where, for simplicity, we regard end as a normal tool. The transition weight $w_{i,j}$ from node v_i to node v_j is computed as follows,

$$w_{i,j} = \frac{\mathbb{E}_D \left[\mathbb{1}\left(tool_s = v_i, tool_{s+1} = v_j\right)\right]}{\mathbb{E}_D \left[\mathbb{1}\left(tool_s = v_i\right)\right]} \quad (5)$$

The $start$ node in $\mathcal{G}$ is set up separately. It connects all tools except the end with equal weights. When tools become massive, a tool retriever based on semantic similarity search can be adopted to select the first one, $tool_1$, as mentioned in Section 3.

3.3 Dynamic Construction

Tools are dynamically changing. They have life cycles and may no longer be maintained by developers. Consequently, the edge weights of $\mathcal{G}$ need timely finetuning, making static construction inapplicable. Moreover, there is generally a lack of large amounts of tool-use trajectories, especially for the emergent plugins on ChatGPT. In this case, dynamic construction of $\mathcal{G}$ is demanding.

$\mathcal{G}$ can be initialized either by static construction or as a non-informative graph with fully connected tools. Then, dynamic construction iterates between generating tool-use trajectories and updating $\mathcal{G}$. The process of trajectory generation is the same as explained in Section 3. The update of $\mathcal{G}$ is important. Since the number of tool-use trajectories is limited, a fine-grained inspection of trajectories is needed. An LLM acts as a tool evaluator, taking the whole trajectory as input and scoring every tool used. These scores are discrete integers in $[-3, 3]$. Evaluating the tools equates to assessing the nodes visited in $\mathcal{G}$. We use $\Delta_i^{(n)}$ to denote the Evaluator's score of node v_i in the n-th iteration. Let $s_i^{(n)}$ denote the accumulated score, e.g., $s_i^{(n)} = \sum_{k=1}^{n} \Delta_i^{(k)}$. The transition weight $w_{i,j}^{(n)}$ is updated as follows.

$$w_{i,j}^{(n)} = \beta w_{i,j}^{(0)} + (1-\beta)\Delta w_{i,j}^{(n)}, \quad (6)$$

$$\Delta w_{i,j}^{(n)} := \frac{f(s_i^{(n)})}{\sum_{v_j \in oneigh(v_i)} f(s_j^{(n)})}, \quad (7)$$

$$f(x) := \begin{cases} \alpha x + 1, & x \geq 0 \\ e^{\alpha x}, & x < 0 \end{cases}, \quad (8)$$

where $f(x)$ maps accumulated scores to $(0, +\infty)$. $\Delta w_{i,j}^{(n)} \in (0, 1]$ denotes the normalized gradient to update the transition weight. α and β are hyperparameters. α controls the speed of updating. β interpolates between the prior weight $w_{i,j}^{(0)}$ and the weight learnt from dynamic construction.

4 Experiments

4.1 Setup

Datasets. We conduct experiments on five datasets: (1) **SciQA** (Lu et al., 2022a), a question-answering benchmark in various scientific fields including multiple subjects like biology, geography, and ecosystems. (2) **TabMWP** (Lu et al., 2022b), a question-answering benchmark, where an LLM should answer questions according to a given Table. (3) **MATH** (Hendrycks et al., 2021), a question answering benchmark mathematical fields including 7 categories: Prealgebra, Algebra, Number Theory, Counting and Probability, Geometry, Intermediate Algebra, and Precalculus. (4) **APIBank** (Li et al., 2023), a multi-task benchmark consisting of various tools to evaluate the performance of tool-augmented LLMs. (5) **ToolBench** (Qin et al., 2023), a large-scale benchmark with 3451 APIs spanning distinct domains. Due to cost considerations, we select subsets of these datasets for evaluation. From the SciQA dataset, we randomly select 1,000 questions for evaluation. For TabMWP, we utilize the test1k subset. Within the MATH dataset, we exclusively consider level-5 questions, which represent the highest difficulty level. Additionally, we exclude geometry-related questions, aligning with prior research (Drori et al., 2022; Wu et al., 2023). For APIBank, we focus on a specific subset, specifically the test data falling within the lv1-lv2-samples category, totaling 212 questions. In the ToolBench dataset, we select a random subset of 1,000 questions from the G1 set.

Baselines. (1) **ReAct** (Yao et al., 2022). Equipped with various tools, an LLM strategically chooses one tool at a time for each step to address specific tasks or questions, ensuring a step-by-step, focused approach to problem-solving. (2) **Reflexion** (Shinn et al., 2023)). Similar to ReAct, LLM interacts with multiple tools in several steps. When an LLM fails a task, it will be prompted to verbally reflect on its reasoning trajectories. The self-refined LLM can learn from the prior errors and retry until the correct answer is submitted or the maximum tries are reached. (3) **Tree-of-Thought** (ToT) (Yao et al., 2023). It formulates the LLM reasoning process as a tree of thoughts, where each node within the tree symbolizes a segment of the solution. At any given node, a thought generator module is responsible for the creation of new nodes, effectively branching out the solution space. Subsequently, each of these new nodes undergoes evaluation. The progression and expansion of this thought tree are governed by a specific search algorithm, such as depth-first search (DFS), which dictates the sequence and manner the nodes are extended and explored. **Implementation details.** We use *gpt-3.5-turbo* as the LLM in all our experiments. Given a question or task instruction, the LLM iteratively selects one from the available tools till the *Answer/Finish* tool is invoked to submit their answer or claim task failure when a maximum iteration of 8 is reached.

Metrics. Evaluation is carried out in the aspects of answer quality and token consumption. Exact match (EM) is adopted to compare the answers generated from LLMs with the groundtruth provided in SciQA, TabMWP, MATH, and APIBank. There is no unique groundtruth answer in ToolBench, *win rate* is therefore adopted. It measures the proportion of answers identified as correct by an external

Table 2: A list of representative tools utilized in three task-specific QA datasets. ✗ denotes tools are irrelevant or noisy to the dataset.

Tool	Description	Dataset		
		SciQA	TabMWP	MATH
GoogleSearch	Search by google	✓	✗	✗
WikiPediaSearch	Search in Wikipedia	✓	✗	✗
ExecuteCode	Execute math expressions	✗	✓	✓
RunPython	Run python code	✗	✓	✓
Search*	Return 'Nothing Found'	✗	✗	✗
LoopUp*	Return 'Nothing Found'	✗	✗	✗
Calculator*	Return random numbers	✗	✗	✗

* means this tool is on purpose designed to be noisy.

Table 3: Results on subsets of close-set task-specific datasets. Noisy and task-irrelevant tools, as described in Table.2, are introduced to analyze the robustness of prompting paradigms. Reflexion is not evaluated on SciQA, a multiple-choice QA dataset, as it would not provide a fair comparison.

Dataset	Size	#APIs	Metric	ReAct	ReAct (clean)	Reflexion	ToT	ToolNet (**Ours**)
SciQA	1000	16	EM	0.45	0.48	-	0.12	**0.61**
			#Tokens	8270	7318	-	23070	5945
			#Steps	5.8	5.4	-	8.9	4.0
TabMWP	1000	16	EM	0.26	0.50	0.57	0.09	**0.67**
			#Tokens	8936	6594	24710	19669	4062
			#Steps	6.1	5.3	11.3*	8.7	3.7
MATH	675	16	EM	0.13	0.20	**0.29**	0.07	0.25
			#Tokens	9216	8382	28886	18090	6602
			#Steps	6.9	6.8	9.5	8.2	5.4

LLM-powered evaluator. In ToolBench, the built-in ToolEval (powered by *gpt-3.5-turbo*) is adopted to compute the win rate. In addition, we measure the number of tokens consumed by LLMs for task completion. It takes into consideration both the input prompts and the generated tokens for reasoning and answering.

4.2 Results and Analysis

4.2.1 Evaluation on Task-Specific Datasets

In practice, noisy and task-irrelevant tools will inevitably exist when LLMs are connected with massive tools. To simulate this scenario, we first carried out experiments on three task-specific QA datasets, namely, SciQA, TabMWP, and MATH. The LLM, *gpt-3.5-turbo*, is connected with 16 tools, of which most are irrelevant to the task the dataset was designed for. Table 2 lists some typical tools used. To the best of our knowledge, there are no public-available tool-use trajectories on these datasets. Therefore, *ToolNet* creates the tool graph by dynamic construction, starting with a fully-connected graph with equal edge scores.

Table 3 compares existing tool learning methods (ReAct, Relfexion, and ToT) with the proposed *ToolNet*. Compared with Reflexion, *ToolNet* uses only 16.4% and 22.8% tokens on TabMWP and MATH, respectively. In terms of answer quality (measured by EM), *ToolNet* significantly surpasses all the other competing approaches on SciQA and TabMWP, achieving a minimum absolute improvement of 10%. Admittedly, Reflexion is slightly better than *ToolNet* on MATH but suffers from significantly increased token consumption and more reasoning steps. In addition, we study the effect of noisy tools on tool-augmented LLMs, with ReAct (clean) as a showcase. It denotes running ReAct by removing all noisy tools on the three datasets. Compared to ReAct (clean), ReAct suffers from dramatic answer quality degradation, increased token consumption, and longer reasoning steps, suggesting its inefficacy in handling noisy and task-irrelevant tools.

In our analysis of the tool scores $s_i^{(n)}$ generated by *ToolNet*, as illustrated in Figure 3, we observe a clear pattern. For each task-specific dataset, the scores of effective tools show a consistent increase, whereas the scores for irrelevant tools remain approximately zero. This pattern demonstrates *ToolNet*'s proficiency in selecting the most suitable tools for distinct tasks. Specifically, *GoogleSearch* emerges as the preferred tool for the SciQA dataset. In contrast, for the TabMWP and MATH datasets, *ExecuteCode* and *RunPython* are identified as the most appropriate tools, respectively. Additionally, *ToolNet* effectively down-weights noisy tools, further optimizing the tool selection process.

4.2.2 Evaluation on Multi-Task Datasets with Massive Tools

We then evaluate *ToolNet* on APIBank and ToolBench, two challenging datasets that consist of diverse tasks and have significantly large amounts of available tools. The massive tools in ToolBench make LLMs difficult to select the starting tool. We follow the default setup in ToolBench to use Tool Retriever, a fine-tuned BERT model, to recommend the K most proper starting tools to LLM. We set K = 8 to augment ReAct and Reflexion on ToolBench and K = 5 on APIBank. For *ToolNet* and *ToT*, only the most recommended tool is provided as the starting tool. The graph of tools is firstly statically constructed from the trajectories provided in the

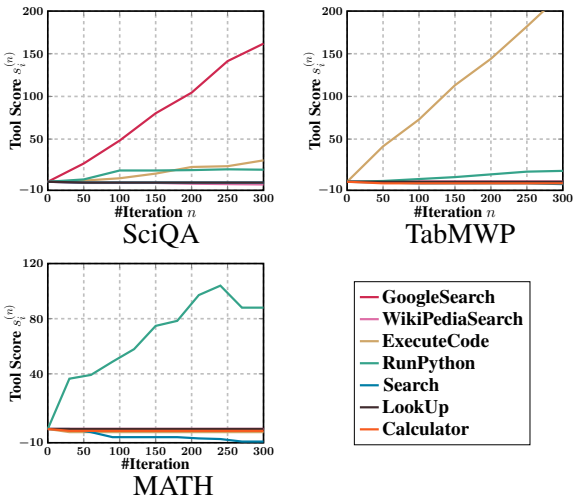


Figure 3: Analysis of tool scores on SciQA, TabMWP, and MATH in dynamic graph construction. Tools effective for each task-specific dataset exhibit increasing scores, indicating *ToolNet*'s capability to automatically select the most appropriate tools for each specific task.

datasets, as is explained in Section 3.2. Thereafter, the graph is updated dynamically at every iteration.

As indicated in Table 4, *ToolNet* is on par with Reflexion in terms of win rate, while utilizing significantly fewer tokens (only 38.5% on APIBank and 49.7% on ToolBench, compared to Reflexion). Additionally, Reflexion's methodology is complementary to *ToolNet*. When integrated into *ToolNet*, it further enhances answer quality, leading to the highest win rates on both datasets.

The large number of tools and the diverse tasks provided by the two datasets prohibit an intuitive interpretation of tool scores and manual tool selection by humans. Meanwhile, tools are alive and constantly evolving. Some tools may occasionally get crashed and can no longer used. This demands *ToolNet* to adaptively raise the sores of some other backup tools. We showcase that dynamic construction can mitigate such a problem. In Figure 4, we simulate on APIBank the scenario where a tool suddenly breaks at the 50th iteration. The score of the broken tool experiences a significant decline whereas the backup tool is swiftly activated and effectively supplants the impaired one. This transition underscores the robust adaptability inherent in *ToolNet*. In this regard, tool redundancy serves as a critical feature in enhancing system reliability.

4.2.3 Effect of Transition Weights

Tools along with the transition weights are formatted as the input contexts to LLMs. The transition

Table 4: Results on multi-task datasets with massive tools. Following the prior work (Qin et al., 2023), a fine-tuned BERT model provided in ToolBench is leveraged to recommend tools for LLMs. ReAct, Reflexion, and ToT are recommended with 8 tools in every reasoning step. *ToolNet* is provided only with one tool in the first step.

Dataset	Size	#APIs	Metric	ReAct	Reflexion	ToT	ToolNet (Ours)	
								+Reflexion
APIBank	212	49	EM	0.70	0.77	0.63	0.75	**0.83**
			#Tokens	1720	4286	2013	**1649**	3353
ToolBench	1000	3992	Win Rate	0.61	0.71	0.60	0.69	**0.75**
			#Tokens	8636	13217	10167	**6575**	12548

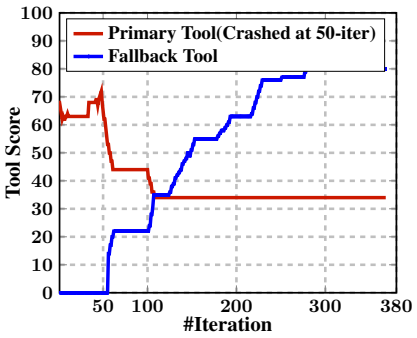


Figure 4: Scores of primary and fallback tools with identical functionality (*GetUserToken* of APIBank) in dynamic graph construction. The primary tool crashed at the 50th iteration. *ToolNet* effectively down-weighs the crashed tool and revives the fallback one, demonstrating its resilience against sudden tool failures.

weights indicate the priorities of tools, thus playing a vital role in tool selection. This section analyses how the transition weights affect the overall performance. As shown in Table 5, five test conditions are considered: removing weights from the input to LLMs (No weight), weights divided by 100 with decimals kept ($/100$), weights divided by 10 with decimals kept ($/10$), removing decimal parts of weights (Integer), and multiplying weights by 10 ($\times 10$). Experiments in each condition were repeated three times to reduce randomness. Worst performance is attained when the weights are removed or scaled down into decimals. This suggests the importance of a proper format of weights. Moreover, there is a general increase in EM when scaling up the transition weights. This suggests that LLMs are more sensitive to the difference between integers or large numbers than the subtle difference between decimals. The token consumption is also gradually reduced when scaling up the weights. These observations may indicate a potential optimization pathway for tool learning. Notably, there

Metric	Modification of Transition Weights				
	No weight	/100	/10	Integer	×10
EM	0.70	0.70	0.71	**0.75**	**0.75**
#Tokens	1710	1682	1664	1649	**1645**

Table 5: How transition weights affect the performance of *ToolNet*. The evaluation is carried out on APIBank.

is an upper-performance limit when scaling up the weights, e.g., the performance saturates at $\times 10$.

5 Related Work

Fine-tuning LLMs to use tools. Early research heavily relied on model fine-tuning for domain-specific tool learning and retrieval-augmented generation. Prominent works in this area include REALM (Guu et al., 2020), RETRO (Borgeaud et al., 2022),VisualGPT (Chen et al., 2022), etc. Notably, WebGPT (Nakano et al.), a GPT-3 fine-tuned on human web search behaviors, can effectively utilize web browsers for question answering. Furthermore, there has been a notable shift in research towards tuning LLMs on a broader spectrum of general tools. Example works include Toolformer (Schick et al., 2023), ToolkenGPT (Hao et al., 2023), Gorilla (Patil et al., 2023), ToolLLM (Qin et al., 2023) etc. However, it is important to acknowledge that collecting tool-use data for fine-tuning can be prohibitively expensive, and these fine-tuned LLMs often struggle to generalize to emergent or updated tools.

Prompting LLMs to use tools. The remarkable in-context learning ability of LLMs motivates prompt engineering approaches for tool learning (Mialon et al., 2023). This is achieved by showing tool descriptions and demonstrations in context. Building on this idea, reasoning chains can be incorporated to tackle more complex problems, such as arithmetic calculation (Cobbe et al., 2021), code execution (Gao et al., 2023), and complex mathematical theory verification (Jiang et al., 2022). More specifically, Plan-and-Solve (Wang et al., 2023) enhances CoT with an explicit planning stage. Self-reflection (Shinn et al., 2023; Madaan et al., 2023; Paul et al., 2023) and self-evaluation (Xie et al., 2023) were introduced recently to self-correct mistakes in reasoning, showing enhanced performance in code generation and computer operation tasks. These paradigms have given rise to popular industry products such as ChatGPT Store and LangChain (Chase, 2023), as well as pioneering experimental products such as AutoGPT (Richards, 2023), MetaGPT (Hong et al., 2023), etc. Furthermore, by calling tools to interact with the virtual or physical world, LLMs are capable of guiding embodied agents to accomplish various household tasks (Huang et al., 2022a,b; Singh et al., 2023). Recent studies utilize LLMs as the central controller to coordinate multiple neural models, achieving promising results in multi-modal reasoning tasks (Lu et al., 2023; Shen et al., 2023). Nevertheless, all methods based on in-context learning suffer from inferior performance in complex scenarios, where the tools are unfamiliar or numerous.

6 Conclusion

We introduce *ToolNet*, a novel plug-and-play method to organize massive tools into a directed graph, facilitating their use by LLMs via in-context learning. A key feature of *ToolNet* is its adaptive tool transition weights, which can be initially set and continually updated. This adaptability ensures rapid integration of new tools and alignment with the changing environment of extensive tool repositories. Comprehensive evaluations on both task-specific datasets and complex, multi-task tool learning datasets have shown that *ToolNet* consistently enhances performance and achieves up to 2.6x greater token efficiency. We believe that this research will inspire future studies to develop more advanced tool-augmented LLMs that can intelligently connect massive real-world tools to fulfill diverse human requirements.

7 Limitations and Future Work

While simple and extensible to concurrent tool learning methods, this work has limitations. First, the proposed graph construction demands tool-use trajectories, which can be costly to collect. Second, it requires that the trajectories include multi-hop tool-use cases to model tool transition. However, existing benchmarks are primarily designed to be task-specific and consist mainly of single tool use cases. Third, due to cost considerations, we only use *gpt-3.5-turbo* in our experiments. Exploring more powerful LLMs such as GPT-4 and the open-source Mixtrial model (Mistral AI Team, 2023) has the potential to further improve performance.

References

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.

Harrison Chase. 2023. LangChain: Next Generation Language Processing.

Jun Chen, Han Guo, Kai Yi, Boyang Li, and Mohamed Elhoseiny. 2022. Visualgpt: Data-efficient adaptation of pretrained language models for image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18030–18040.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

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. 2023. Metagpt: Meta programming for multi-agent collaborative framework.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

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

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

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

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2022b. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *The Eleventh International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

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

Microsoft Corporation. 2023a. Microsoft copilot. Accessed: 2023-12-13.

Microsoft Corporation. 2023b. New bing. https://www.bing.com/new. Accessed: 2023-12-13.

Mistral AI Team. 2023. Mixtral of experts. Accessed: 2023-12-14.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: browser-assisted question-answering with human feedback (2021). *URL https://arxiv.org/abs/2112.09332*.

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

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*.

Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, Cuiyun Gao, and Michael R Lyu. 2022. Revisiting, benchmarking and exploring api recommendation: How far are we? *IEEE Transactions on Software Engineering*, 49(4):1876–1897.

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

Toran Bruce Richards. 2023. Auto-GPT: An Autonomous GPT-4 Experiment.

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*.

Zhenwei Shao, Zhou Yu, Meng Wang, and Jun Yu. 2023. Prompting large language models with answer heuristics for knowledge-based visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14974–14983.

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

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

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

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. doi: 10.48550. *arXiv preprint arXiv.2306.05301*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Yiran Wu, Feiran Jia, Shaokun Zhang, Qingyun Wu, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, and Chi Wang. 2023. An empirical study on challenging math problem solving with gpt-4. *arXiv preprint arXiv:2306.01337*.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Decomposition enhances reasoning via self-evaluation guided decoding. *arXiv preprint arXiv:2305.00633*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.