Toolken+: Improving LLM Tool Usage with Reranking and a Reject Option

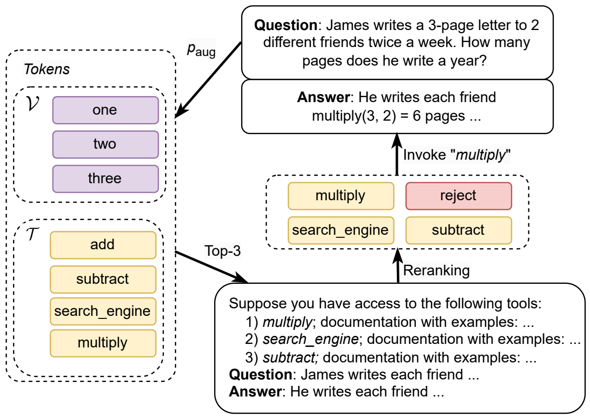
Konstantin Yakovlev[1]*, **Sergey Nikolenko**[2,3], **Andrey Bout**[4]*

[1]HSE University, Moscow, Russia
[2]ISP RAS Research Center for Trusted Artificial Intelligence, Moscow, Russia,
[3]St. Petersburg Department of the Steklov Institute of Mathematics, Russia,
[4]Yandex, Moscow, Russia

Correspondence: kdyakovlev@hse.ru, sergey@logic.pdmi.ras.ru, andrey-bout@yandex-team.ru

Abstract

The recently proposed ToolkenGPT tool learning paradigm demonstrates promising performance but suffers from two major issues: first, it cannot benefit from tool documentation, and second, it often makes mistakes in whether to use a tool at all. We introduce Toolken+ that mitigates the first problem by reranking top k tools selected by ToolkenGPT and the second problem with a special "Reject" option such that the model will generate a vocabulary token if "Reject" is ranked first. We demonstrate the effectiveness of Toolken+ on multistep numerical reasoning and tool selection tasks.



Question: James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?

Answer: He writes each friend multiply(3, 2) = 6 pages ...

Suppose you have access to the following tools:
1) *multiply*; documentation with examples: ...
2) *search_engine*; documentation with examples: ...
3) *subtract;* documentation with examples: ...
Question: James writes each friend ...
Answer: He writes each friend ...

Figure 1: Toolken+ sample operation.

1 Introduction

Recently, large language models (LLM) have been extended by allowing access to external tools such as symbolic computation engines (Gou et al., 2023b), databases that serve as external memory (Mu et al., 2023), and others (Schick et al., 2023; Qin et al., 2023). Tool learning paradigms can be broadly divided into (1) *supervised fine-tuning* to leverage tools (Schick et al., 2023), which works well but lacks flexibility and cannot generalize to unseen tools, and (2) *in-context learning* (Lu et al., 2023), where demonstrations are provided in the prompt; this method is very easy to extend and generalize but often bumps against inherent context length limitations of LLMs (Tay et al., 2021). *ToolkenGPT* (Hao et al., 2023) aims to have the best of both worlds: each tool is represented by a special token called *toolken* that has a trainable embedding and extends the vocabulary. Once a toolken has been predicted, the model switches into "tool mode" where it uses in-context examples to fill in the tool's arguments, calls it, sends the results back to text, and returns to language modeling mode. Toolkens require very few parameters (toolken embeddings) to be trained while keeping LLM weights frozen, and there is no limit on the amount of data to train these parameters.

In this work, we extend ToolkenGPT with two novel features that aim to fix two important issues. First, ToolkenGPT cannot use tool documentation known to be helpful for LLMs (Hsieh et al., 2023); we will show that ToolkenGPT is often unsure which tool to use, and documentation could help decide this. To this end, we introduce a copy of toolken embeddings that rerank retrieved tools, i.e., take top-k tool candidates, prepend the prompt with their documentations, and ask the LLM to choose the most relevant tool. Second, ToolkenGPT often makes mistakes in judging when to use tools, calling them too often. To alleviate this, we introduce an extra REJ ("*Reject*") "tool" that switches back to text generation without invoking any tools. REJ is provided as an option for the reranking mechanism introduced above. The entire operation of Toolken+ is illustrated in Figure 1.

Overall, we aim to minimize false positive errors for tool invocations and tool misclassification rate for ToolkenGPT. This significantly improves the model's robustness, allowing for developing more trusted LLM agents that can have access to a wider variety of tools. A general tool usage paradigm for

* Work was done while at Huawei Noah's Ark Lab

LLMs (Yang et al., 2023; Huang et al., 2023b) has four stages: whether to use a tool, which tool to use, infilling the arguments, and dealing with the tool's output. Our methods improve the first and second stages of this process. Moreover, we provide a formal justification for the toolken training algorithm based on variational inference. We evaluate our results on the GSM8K (Cobbe et al., 2021), MetaTool (Huang et al., 2023b), and VirtualHome (Puig et al., 2018) datasets, showing significant improvements.

Thus, our contributions are as follows: (1) solutions to issues associated with the first two stages of the tool usage process for LLMs; (2) a theoretically grounded training procedure for the introduced toolken embeddings; (3) an empirical evaluation study supporting the efficiency of our approach. Below, Section 2 compares our approach with recent related work, Section 3 introduces our modifications and formal justification for training and inference, Section 4 presents experimental results, and Section 5 concludes the paper.

2 Related Work

Quantifying the uncertainty of LLMs. Recently, Zhang et al. (2023) introduced a tuning method to teach LLMs to refrain from answering the question if the LLM is not sure in its answer, reducing hallucinations and improving uncertainty estimation. In contrast to this study, our approach does not require any fine-tuning of the LLM, instead we learn an embedding corresponding to the rejection tool. In another line of research, Diao et al. (2023) proposed active prompting focusing on finding the best task-specific prompt. In this work, we focus on task-independent prompts.

Natural language feedback. Huang et al. (2023a) suggested the idea to use LLM feedback as training inputs, using the generated rationale-augmented answers to fine-tune the model. To alleviate incorrect reasoning steps with tool usage, Paul et al. (2023) suggested to generate natural language feedback from a critic model learned separately. Note that although this work also addresses the issue of using incorrect operations in the Math World Problem task, it requires to train a critic model.

Overall, our approach adheres to the paradigm of prompted refiners as shown in (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2023a), where the same frozen model is used for reasoning and providing feedback. Our method does not invoke the tools themselves and does not work with their outputs, only produces special tokens for tool invocation. In a recent work, An et al. (2024) also used model mistake correction to improve the quality of solving math problems. However, their suggested approach relied on GPT-4 output and, again, needed to fine-tune the model.

Chain of thought reasoning. Chain of thought prompting was originally introduced by Wei et al. (2023); Zhou et al. (2022) to enhance the ability of large language models to perform complex reasoning. Kojima et al. (2022) showed that a LLM is a good zero-shot reasoner with a simple prompt before each answer. To further improve reasoning skills, Wang et al. (2022) introduced self-consistency decoding that reranks the generated rationales by taking a majority vote over the final numerical answers. Similarly, our approach also benefits from reranking that can use additional information such as tool documentation.

Recently, Zelikman et al. (2022) proposed a bootstrapping technique that was able to improve the performance on reasoning tasks even without a massive rationale dataset. In their approach, the model is repeatedly fine-tuned on a dataset of self-generated rationales; our approach is similar to this one since bootstrapping is used to train the introduced toolkens but we do not need to tune the LLM weights.

Tool-augmented language models. One direction for augmenting LLMs with external tools is fine-tuning for tool use (Qin et al., 2023; Liang et al., 2023; Schick et al., 2023; Patil et al., 2023); these methods achieve excellent performance but suffer from poor adaptability to unseen tools and high computational requirements to fine-tune the LLM.

Another paradigm learns tool use in context, from documentation and/or demonstrations added to the LLM input (Lu et al., 2023; Paranjape et al., 2023; Shen et al., 2023); these approaches do not require fine-tuning and can learn a new tool given a handful of demonstrations but suffer from performance degradation because of limited context length.

In this work, we propose an improvement for the Toolken paradigm recently proposed by Hao et al. (2023) that aims to take the best of both worlds. It introduces learnable tool embeddings (toolkens) trained on a dataset of tool use examples while keeping the weights of the LLM frozen; it also leverages in-context learning to fill in tool arguments.

3 Method

ToolkenGPT. ToolkenGPT introduces an embedding for each tool concatenated with the language modeling head. Formally, the embedding matrix $\mathbf{W}_\mathcal{V} \in \mathbb{R}^{|\mathcal{V}|\times d}$, where $\mathcal{V}$ is the original vocabulary of tokens and d is the latent dimension, is extended with a matrix $\mathbf{W}_\mathcal{T} \in \mathbb{R}^{|\mathcal{T}|\times d}$ for a set of tools $\mathcal{T} = \{t_1, \dots, t_{|\mathcal{T}|}\}$, and the next token probability for the augmented LLM is calculated as $p_{\text{aug}}(\mathbf{x}_i|\mathbf{x}_{<i}) = \operatorname{softmax}([\mathbf{W}_\mathcal{V}, \mathbf{W}_\mathcal{T}]\,\mathbf{h}_{i-1})$. Inference is divided into two interleaving stages: reasoning mode, where the model generates a rationale using $p_{\text{aug}}(\mathbf{x}_i|\mathbf{x}_{<i})$, and tool mode, where it infills tool arguments given a prompt with usage examples. Thus, $\mathbf{W}_\mathcal{T}$ are learned by solving

$$\min_{\mathbf{W}_\mathcal{T}} \sum_{X\in D} \sum_{i=1}^{|X|} -\log p_{\text{aug}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}). \quad (1)$$

Toolken+. First, we extend the tool set as $\mathcal{T}' = \mathcal{T} \cup \{\text{REJ}\}$, where REJ is a special tool responsible for switching back to reasoning mode. Second, instead of taking the best proposed tool we take the retrieved subset of top k tools $\mathcal{T}_k \subseteq \mathcal{T}'$ and ask the model to choose one. The proposed Toolken+ model takes the previously generated sequence and top k tools retrieved by ToolkenGPT $\mathcal{T}_k$ as input and is asked to generate a tool from $\mathcal{T}_k \cup \{\text{REJ}\}$. Formally, Toolken+ produces $p_{\text{rank}}(\mathbf{x}_i|\mathbf{x}_{<i}, \mathcal{T}_k) = \operatorname{softmax}(\mathbf{W}_{\mathcal{T}'}\mathbf{h}_{i-1} + \mathbf{m}(\mathcal{T}_k))$, where $\mathbf{m}(\mathcal{T}_k) \in \mathbb{R}^{|\mathcal{T}'|}$ is the mask vector with $\mathbf{m}(\mathcal{T}_k)_t = 0$ for $t \in \mathcal{T}_k \cup \{\text{REJ}\}$ and $-\infty$ otherwise. Toolken+'s inference procedure is shown in Algorithm 1.

Approximate inference. The next token probabilities in Algorithm 1 are given by

$$\begin{aligned} p(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}) &= [\![\mathbf{x}_{i+1} \in \mathcal{V}]\!]\,(p_{\text{aug}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}) \\ &+ p_{\text{aug}}(\mathcal{T}|\mathbf{x}_{\le i})\,p_{\text{rank}}(\text{REJ}|\mathbf{x}_{\le i}, \mathcal{T}_k)\,p_{\text{LLM}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})) \\ &+ [\![\mathbf{x}_{i+1} \in \mathcal{T}]\!]\,p_{\text{aug}}(\mathcal{T}|\mathbf{x}_{\le i})\,p_{\text{rank}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}, \mathcal{T}_k), \end{aligned}$$

where $[\![\cdot]\!]$ is the indicator. For a dataset $D = \{X_n\}_{n=1}^N$, tool embeddings are found by solving

$$\min_{\mathbf{W}_\mathcal{T}, \mathbf{W}_{\mathcal{T}'}} \sum_{X\in D} \sum_{i=1}^{|X|} -\log p(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}), \quad (2)$$

which is non-differentiable w.r.t. $\mathbf{W}_\mathcal{T}$, so we optimize the original ToolkenGPT model with its own criterion (1) and then optimize (2) w.r.t. $\mathbf{W}_{\mathcal{T}'}$ only. Problem (2), however, suffers from computational instabilities caused by the product of model probabilities in $p(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})$, so we propose to optimize a computationally stable upper bound instead.

Algorithm 1: Toolken+ inference

```
Data: $p_{\text{LLM}}(\mathbf{x}_i|\mathbf{x}_{<i})$, $p_{\text{aug}}(\mathbf{x}_i|\mathbf{x}_{<i})$,
      $p_{\text{rank}}(\mathbf{x}_i|\mathbf{x}_{<i}, \mathcal{T}_k)$, user query $q$
Result: Rationale $\mathbf{x}_{1:n}$ (with a user query)
$x \leftarrow q$, $i \leftarrow |q|$;
while $\mathbf{x}_i \ne EOS$ do
    $\mathbf{x}_{i+1}^{(0)} \sim p_{\text{aug}}(\cdot|\mathbf{x}_{\le i})$;
    if $\mathbf{x}_{i+1}^{(0)} \in \mathcal{T}$ then
        $\mathcal{T}_k \leftarrow \text{TopkTools}(p_{\text{aug}}(\cdot|\mathbf{x}_{\le i}))$;
        $\mathbf{x}_{i+1}^{(1)} \sim p_{\text{rank}}(\cdot|\mathbf{x}_{\le i}, \mathcal{T}_k)$;
        if $\mathbf{x}_{i+1}^{(1)} = \text{REJ}$ then $\mathbf{x}_{i+1} \sim p_{\text{LLM}}(\cdot|\mathbf{x}_{\le i})$
          else $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1}^{(1)}$ ;
    else
        $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1}^{(0)}$;
    $i \leftarrow i + 1$;
```

Proposition 1 (Naive upper bound). *The following is a computationally stable upper bound of* (2)*, up to an additive constant independent of* $\mathbf{W}_{\mathcal{T}'}$*:*

$$\begin{aligned} \sum_{X\in D} \sum_{i=1}^{|X|} (&- [\![\mathbf{x}_{i+1} \in \mathcal{V}]\!] \log p_{\text{rank}}(\text{REJ}|\mathbf{x}_{\le i}, \mathcal{T}_k) \\ &- [\![\mathbf{x}_{i+1} \in \mathcal{T}]\!] \log p_{\text{rank}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}, \mathcal{T}_k)). \quad (3) \end{aligned}$$

Proof. We transform (omitting $\mathcal{T}_k$ for brevity)

$$\begin{aligned} &\log p(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}) = \log p(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})\,([\![\mathbf{x}_{i+1} \in \mathcal{V}]\!] + \\ &+ [\![\mathbf{x}_{i+1} \in \mathcal{T}]\!]) \ge [\![\mathbf{x}_{i+1} \in \mathcal{V}]\!]\,(\log p_{\text{rank}}(\text{REJ}|\mathbf{x}_{\le i}) + \\ &+ \log p_{\text{aug}}(\mathcal{T}|\mathbf{x}_{\le i}) + \log p_{\text{LLM}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})) + \\ &+ [\![\mathbf{x}_{i+1} \in \mathcal{T}]\!]\,(\log p_{\text{aug}}(\mathcal{T}|\mathbf{x}_{\le i}) + \log p_{\text{rank}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})), \end{aligned}$$

where the inequality holds because $p_{\text{aug}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i})$ is always nonnegative, and then obtain (3) by removing the terms independent of $\mathbf{W}_{\mathcal{T}'}$. □

Bound (3) is differentiable but still computationally hard: it requires $|X|$ forward passes to find the loss for a single data point, retrieving top k tools for every i. Therefore, we propose a simplified objective that uses only i where ToolkenGPT erroneously predicts a toolken instead of a regular token:

$$\begin{aligned} \min_{\mathbf{W}_{\mathcal{T}'}} \sum_{X\in D} \sum_{i=1}^{|X|} \Bigg(&- [\![\mathbf{x}_{i+1} \in \mathcal{T}]\!] \log p_{\text{rank}}(\mathbf{x}_{i+1}|\mathbf{x}_{\le i}, \mathcal{T}_k) \\ &- \left[\!\!\left[\begin{matrix} \mathbf{x}_{i+1} \in \mathcal{V} \\ \arg\max p_t \in \mathcal{T} \end{matrix}\right]\!\!\right] \log p_{\text{rank}}(\text{REJ}|\mathbf{x}_{\le i}, \mathcal{T}_k) \Bigg). \quad (4) \end{aligned}$$

We train Toolken+ with (4) to correct the errors of ToolkenGPT or rescore its outputs. Objective (4) is not guaranteed to be an upper bound of (1) but can be viewed as an approximation via hard negative mining ("hard" prefixes $\mathbf{x}_{\le i}$ are those where ToolkenGPT incorrectly predicts tool use).

Dataset	Learning rate	Epochs
GSM8K	10^{-4}	5
MetaTool	10^{-4}	1-3
VirtualHome	10^{-3}	5-10

Table 1: Hyperparameters used for training ToolkenGPT and Toolken+.

4 Experiments

Datasets and setup. We evaluate Toolken+ on three datasets. *GSM8K* (Cobbe et al., 2021) is a parallel dataset of math problems and their rationales. We use the multistep reasoning task with four arithmetic operations as tools, removing equations from the rationales except for intermediate results (e.g., "Weng earns 12/60 = 0.2 per minute" becomes "Weng earns 0.2 per minute") to make tool selection harder. *MetaTool* (Huang et al., 2023b) is a parallel dataset of user queries and tools with their descriptions; we use all available tools for tool selection. *VirtualHome* (Puig et al., 2018) is a dataset of complex household activities represented by plans, sequences of verb-object expressions where verbs and objects are external tools. Following Hao et al. (2023), we split it into a training set of 247 tasks and a test set of 50 tasks, using 25 verbs and 32 objects in total. Hyperparameter values and detailed experimental settings are reported in the Appendix: Prompts for Toolken+. We use open source LLMs: Llama2-7B, Llama2-7B-chat (Touvron et al., 2023a,b), Vicuna-7B, Vicuna-13B (Chiang et al., 2023).

The hyperparameters used to train all our models are reported in Table 1. Overall, we used one data point per parameters update and used the same hyperparameters for ToolkenGPT and Toolken+ regardless of the LLM. All models were trained using the Adam optimizer (Kingma and Ba, 2014).

For *MetaTool*, we took single-tool data that contains about 20K samples. We split it into a test split of 2K examples and two folds of about 9K examples each. To construct the training split for the GSM8K dataset, we removed all equations except for intermediate results. Moreover, we also performed the same procedure with released prompts for reasoning mode and tool mode of ToolkenGPT (Hao et al., 2023). Additionally, for the rejection mechanism of Toolken+ we used the processed prompt of the reasoning mode of ToolkenGPT. For *VirtualHome*, we follow the setup of ToolkenGPT (Hao et al., 2023) with the only difference that we split the training data into two folds. Toolken+ reranks top-3 retrieved objects by ToolkenGPT listed in ascending order by relevance. The prompts used for Toolken+ are shown in the Appendix.

LLM	Tool model	MetaTool	GSM8K	VirtualHome	
		Rec@1	Match	Strict	Relaxed
Vicuna-7B	4-shot	-	16.2	0.04	0.2
	ToolkenGPT	0.623	16.9	**0.62**	0.72
	Toolken+	**0.643**	**18.8**	0.48	**0.74**
Vicuna-13B	4-shot	-	17.8	0.16	0.30
	ToolkenGPT	0.646	18.4	0.34	0.54
	Toolken+	**0.662**	**19.1**	**0.58**	**0.66**
Llama2-7B-chat	4-shot	-	12.7	0.08	0.24
	ToolkenGPT	0.642	11.7	0.44	**0.66**
	Toolken+	**0.692**	**12.8**	**0.56**	**0.66**
Llama2-13B-chat	4-shot	-	**10.3**	0.18	0.26
	ToolkenGPT	0.704	8.8	0.18	0.20
	Toolken+	**0.733**	9.6	**0.54**	**0.58**

Table 2: Experimental results on the *MetaTool*, GSM8K, and *VirtualHome* datasets.

Tool selection. In *MetaTool*, the task is to retrieve a single tool given a query, so there is no need to use the proposed REJ tool, and we use this task to validate the idea of reranking ToolkenGPT's outputs in isolation. Thus, we remove the first term from the loss function (4). We split the training set into two folds, train ToolkenGPT on the first fold, and train Toolken+ on the second fold given the top 5 outputs of ToolkenGPT. We compare both models on a held-out set and report Recall@1 in Table 2. There is no few-shot version for *MetaTool* since 200 tool descriptions do not fit into a prompt. We see that Toolken+ significantly improves the results across all considered LLMs.

Multistep reasoning. In the GSM8K dataset, the task is to generate a numerical answer to a question using the four basic arithmetic operations as tools. Here we demonstrate that the rejection mechanism improves the resulting accuracy even in isolation, so in this setup $k = 1$. We again split the data into two folds, train ToolkenGPT on one fold, collect the errors made on the second fold and train Toolken+ with an extended set of tools. Note that since $k = 1$ the inference speed for the two models is virtually identical. Table 2 reports exact match results and shows that while both tool models improve over the vanilla LLM prompted with 4 samples from the training set, Toolken+ provides additional performance improvements.

Embodied plan generation. In the *VirtualHome*

Tool model	k	**Recall** **@1**	**@3**	**@5**	**Latency**
ToolkenGPT	1	0.623	0.762	0.814	1.0x
Toolken+	3	0.661	0.762	0.762	2.6x
	5	0.652	0.780	0.814	3.0x
	7	0.653	0.775	0.822	3.3x
	10	0.637	0.802	0.843	4.1x

Table 3: Ablation study on k, Vicuna-7B base model.

Question: Harry slept 9 hours last night. His friend James slept only 2/3 of what Harry slept. How many more hours did Harry sleep than James? **Answer**: 3

ToolkenGPT: Harry slept '<multiply>(9,2/3) = 6' hours less than James slept. James slept '<divide>(9,2/3) = 13.5' hours (2/3 of 9) and Harry slept '<add>(9,6) = 15' hours (9 - 6 = 3). So Harry slept 15 hours and James slept 13.5 hours. #### 15

Toolken+: Harry slept '<multiply>(9,2/3) = 6' hours less than James slept. James slept '<reject>'2/3 of what Harry slept, so he slept 2/3 * 9 = 6 hours. Harry slept 9 hours, so he slept 9 - 6 = 3 hours more than James. #### 3

Question: Geb is 10 less than half the age of Haley. If Haley is 26 years old, how old is Geb? **Answer**: 3

ToolkenGPT: Half the age of Haley is '<divide>(26,2) = 13' years. So Geb is '<add>(10,13) = 23' years old. #### 23

Toolken+: Half the age of Haley is '<divide>(26,2) = 13' years. So Geb is <reject>10 less than that, which means he is '<subtract>(13,10) = 3' years old. #### 3

Table 4: Case study on GSM8K and Vicuna-7B.

dataset, the task is to generate a sequence of actions given a question. We follow the setup of the tool selection task but now generate a contiguous sequence of actions, setting $k = 3$ for Toolken+ and running Toolken+ only on actions that correspond to an object. We compare the proposed approach with ToolkenGPT and report the success rate and its relaxed version (share of plans that pass through a target state) in Table 2; we see that Toolken+ consistently improves over ToolkenGPT.

Ablation study. In the ablation study, we evaluate how performance depends on the number of tools to be reranked by Toolken+. Table 3 shows the results evaluated on the *MetaTool* dataset with the Vicuna-7B base model; we prepend tool descriptions in the order ranked by ToolkenGPT. We see that performance reaches its optimal value at $k = 3$ and degrades with increasing k.

Case study. Table 4 illustrates the difference in reasoning between ToolkenGPT and Toolken+ with specific examples from GSM8K (we used Vicuna-7B). The reported examples show how the rejection mechanism can allow to prevent the LLM from confusing arithmetic operations calls, which would otherwise lead to an incorrect answer.

5 Conclusion

In this work, we have proposed an improvement for the ToolkenGPT approach of learning special token embeddings that adds a reject option and a reranking mechanism for tool selection. Our approach significantly improves the results via in-context learning while still keeping LLM weights frozen and learning only toolken embeddings.

The considered approach is an important step towards improving the robustness of AI agents and user-facing tools based on modern LLMs. Better tool use not only makes the tool usage results more robust but also has a potential to reduce hallucinations and make LLM answers more trustworthy by allowing an LLM to reliably run external tools to verify its answer; this is a very important consideration in practical usage.

In future work, we hope to extend the Toolken+ approach to other external tools and/or agents to further expand the capabilities of modern LLMs.

6 Limitations

One important limitation of this work is that the LLM used for Toolken+ should be aware of the tools retrieved by ToolkenGPT in the sense that its tool retrieval accuracy should be sufficiently high. The LLM also should be able to improve the ranking of these tools by reading their descriptions: Toolken+ relies on the accuracy of this reranking but it is out of our hands. Another limitation is that Toolken+ has only been evaluated on a limited number of tasks. To make the results more convincing, the framework should be tested on a wide range of tool-learning tasks and datasets.

Acknowledgements

This work was supported by a grant for research centers in the field of artificial intelligence, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002) and the agreement with the Ivannikov Institute for System Programming of the Russian Academy of Sciences dated November 2, 2021 No. 70-2021-00142.

References

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2024. Learning from mistakes makes llm better reasoner.

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

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*.

Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. 2023. Active prompting with chain-of-thought for large language models.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023a. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023b. Tora: A tool-integrated reasoning agent for mathematical problem solving.

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

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023a. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore. Association for Computational Linguistics.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023b. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

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*.

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*.

Yongyu Mu, Abudurexiti Reheman, Zhiquan Cao, Yuchun Fan, Bei Li, Yinqiao Li, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2023. Augmenting large language model translators via translation memories. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10287–10299, Toronto, Canada. Association for Computational Linguistics.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh 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*.

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*.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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*.

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.

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, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

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

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Hanning Zhang, Shizhe Diao, Yong Lin, Yi R Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. 2023. R-tuning: Teaching large language models to refuse unknown questions. *arXiv preprint arXiv:2311.09677*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Appendix: Prompts for Toolken+

Prompt for Vicuna-7B and Vicuna-13B:

```
[System]
Below is the instruction that describes a task.
Write a response using the API tools that
appropriately completes the request.
Your output should follow this format:
Action: API call

[Question]
[QUESTION]

[The Start of Assistant's Answer]
Action:
```

Prompt for Llama2-7B-chat and Llama2-13B-chat

```
<<SYS>>
Below is the instruction that describes a task.
Write a response using the API tools that
appropriately completes the request.
Your output should follow this format:
Action: API call
<</SYS>>

[INST]
[QUESTION]
[/INST]
Action:
```

Prompt for Vicuna-7B and Vicuna-13B:

```
[System]
Suppose you have access to
the following API tools:
1. tool name: [NAME],
tool description: [DESCRIPTION],
example question: [EXAMPLE].
....
Below is the instruction that describes a task.
Write a response using the API tools that
appropriately completes the request.
Your output should follow this format:
Action:

[User Question]
[QUESTION]

[The Start of Assistant's Answer]
Action: $
```

Prompt for Llama2-7B-chat and Llama2-13B-chat:

```
<<SYS>>
Suppose you have access to
the following API tools:
1. tool name: [NAME],
tool description: [DESCRIPTION],
example question: [EXAMPLE].
....
Below is the instruction that describes a task.
Write a response using the API tools that
appropriately completes the request.
Your output should follow this format:

Action: API call
<</SYS>>

[INST]
[QUESTION]
[/INST]
Action:
```

Empirically we found that the tools selected by ToolkenGPT should appear in descending order of relevance.

```
Task 1:
...
Task 4:
I am in [ROOM]. The objects I can manipulate
are [OBJECTS].
Goal:
[GOAL]
Hint:
[HINT]
Plan:
Which of the objects: <obj1>, <obj2>, <obj3>
is best to continue the plan?
[PLAN]
```