GEAR: Augmenting Language Models with Generalizable and Efficient Tool Resolution

Yining Lu[♡] and **Haoping Yu**[♡] and **Daniel Khashabi**
Johns Hopkins University, Baltimore, MD
{ylu130, hyu90, danielk}@jhu.edu

Abstract

Augmenting large language models (LLM) to use external tools enhances their performance across a variety of tasks. However, prior works over-rely on task-specific demonstration of tool use that limits their generalizability and computational cost due to making many calls to large-scale LLMs. We introduce GEAR, a computationally efficient query-tool grounding algorithm that is generalizable to various tasks that require tool use while not relying on task-specific demonstrations. GEAR achieves better efficiency by delegating tool grounding and execution to small language models (SLM) and LLM, respectively; while leveraging semantic and pattern-based evaluation at both question and answer levels for generalizable tool grounding. We evaluate GEAR on 14 datasets across 6 downstream tasks, demonstrating its strong generalizability to novel tasks, tools and different SLMs. Despite offering more efficiency, GEAR achieves higher precision in tool grounding compared to prior strategies using LLM prompting, thus improving downstream accuracy at a reduced computational cost. For example, we demonstrate that GEAR-augmented GPT-J and GPT-3 outperform counterpart tool-augmented baselines because of better tool use.

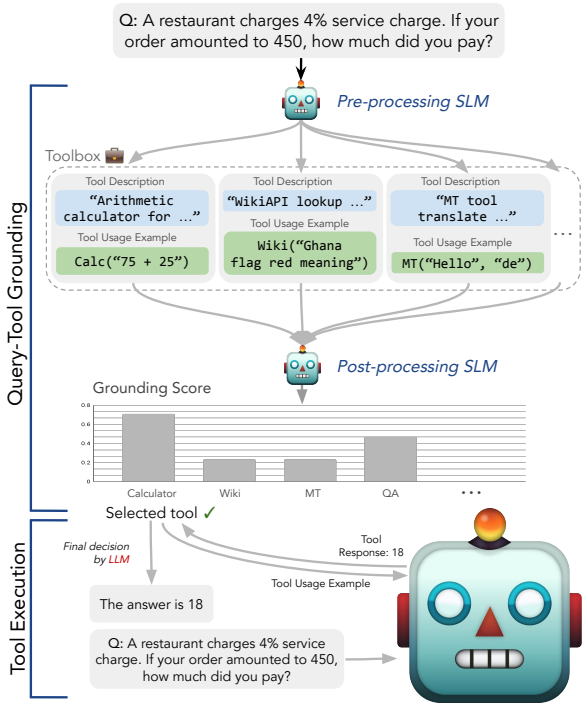


Figure 1: GEAR leverages small language models (SLM) to facilitate the process of *tool grounding* for a given query and has the ability to add and utilize new tools for novel tasks without the need for fine-tuning or extra demonstrations. GEAR utilizes a large language model (LLM) in the *tool execution* module to ensure the accuracy of the final answer.

1 Introduction

Recently there has been a surge in research on Augmented Language Model (Mialon et al., 2023), which aims to enable models interface existing "tools" for various purposes, such as accessing the latest information (Izacard et al., 2022), interacting with third-party services (Liang et al., 2023), performing precise calculations (Schick et al., 2023), or reasoning via code (Cheng et al., 2022; Gao et al., 2022). The paradigmatic framework of these tool-augmented LM studies generally comprises two steps: selecting a tool and executing it via a generated API call. Consequently, choosing suitable tools is essential for task success.

The existing works teach language models to select tools using either fine-tuning or in-context learning approaches. For example, Toolformer (Schick et al., 2023) is tailored and limited to a predetermined set of tools observed during pre-training. On the other hand, approaches based on in-context learning (Li et al., 2023; Paranjape et al., 2023; Chen et al., 2023; Sun et al., 2023; Yao et al., 2022) rely on many calls to LLM and task-specific demonstrations which diminish their cost efficiency and limits their scalability to a large tool library. To address these limitations, we focus on making the query-tool grounding process more *efficient*, *scalable* and *generalizable*.

[♡] Equal contribution

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 112–138

Feature	CoT	Zero-shot CoT	Toolformer	ToolkenGPT	ART	GEAR
Tool Use	✗	✗	✔	✔	✔	✔
Novel Task Generalization	✗	✔	✔	✗	✗	✔
Extensibility to New Tools at Inference	*N/A*	*N/A*	✗	✗	✔	✔
Grounding Algorithm	*N/A*	*N/A*	Finetune	LLM Generation	LLM-Based or Cosine Similarity	GEAR
# of LLM Calls at Inference	1	1	1	1	N	1
Input Data	Task-Specific Demonstrations	Single Query	Augmented Dataset	Supervised Data	Task-Specific Demonstrations	Single Query

Table 1: Comparing GEAR with the recent related works for generalization, computation efficiency, and key grounding algorithms. N is the task library size.

In this work, we present GEAR, **A**ugment language models with **G**eneralizable and **E**fficient tool **R**esolution, a query-tool grounding algorithm that enables efficient use of tools while also allowing for generalization to both new tasks and large tool libraries. The GEAR framework (Figure 1) is comprised of two key modules: (i) Query-Tool Grounding and (ii) Execution. In the *query-tool grounding* module, we compute a grounding score comprised of semantic and pattern based evaluations (introduced in §3). The intuition behind the grounding score is to enable comprehensive query-to-query and answer-to-answer comparisons by leveraging tool description and usage examples, respectively. By considering both question and answer perspectives, the final grounding score provides a comprehensive evaluation of the suitability and compatibility between the given queries and the available tools. Then GEAR passes the selected tool and the given query to the *execution* module where a LLM is prompted to generate the appropriate API call to obtain the ultimate response from the tool. In general, given n tools in a tool library, GEAR makes $(n+1)$ calls to SLMs and only 1 call to LLM (Algorithm 1).

Compared to all other in-context learning approaches (Li et al., 2023; Paranjape et al., 2023), GEAR significantly reduces the workload on the LLM to do tool grounding, subtask decomposition and API call generation across all tools by assigning query-tool grounding to SLM. For instance, compared to ART (Paranjape et al., 2023), GEAR reduces the calls to LLM by directing its intermediate calls to an SLM (e.g., `GPT-Neo`) leading to $4\times$ reduction in computational cost (FLOPS), while providing higher accuracy (details in §5.2; Table 5).

To the best of our knowledge, there is currently no fine-grained algorithm for query-tool grounding, nor have there been comprehensive empirical experiments to assess tool grounding accuracy across various tool library sizes. Thus, we conduct experiments[1] for GEAR on a variety of different downstream tasks and tool libraries. Our experiments demonstrate that, GEAR improves grounding questions to tools, which leads to stronger downstream performance compared to other few-shot or tool-augmented baselines. For example, GEAR leveraging SLMs (e.g., `GPT-Neo` with 1.3B parameters) consistently achieves high grounding performance on 12 datasets from 6 NLP tasks, resulting in better downstream accuracy than few-shot prompting and ART (Paranjape et al., 2023). We also provide evidence of the strong generalizability of GEAR to novel tasks, large tool libraries, and different SLMs.

2 Related Work

We divide the notable prior works on tool-augmented models into two groups based on how they modify language models: one uses fine-tuning, while the other uses in-context prompting. We also touch upon works in embodied LM applications.

Tool Use via Fine-tuning. There have been some research efforts focusing on training models to use various language tools (Thoppilan et al., 2022; Komeili et al., 2022; Shuster et al., 2022; Khot et al., 2021, 2022).

More recently, Schick et al. (2023) proposes Toolformer which uses a self-supervision manner to train LLMs to use Wikipedia, QA, Calculator, Machine Translation, and Calendar tools. Parisi et al. (2022) uses a similar self-supervised approach for teaching models to use tools. Hao et al. (2023) treats tools as special tokens of LLM and learns embeddings for them. Qiao et al. (2023) proposes a two-stage framework that enables the model to learn through feedback derived from tool execution. Yang et al. (2023) employs instruction tuning

[1] Code to reproduce our results is available.

to enable LLMs to use multimodal tools. Although fine-tuning allows somewhat accurate tool grounding among those observed during training, a key issue with the resulting models is that they cannot utilize new tools without retraining, thus hindering models' generalizability to new tools and tasks.

Tool Use via In-Context Learning. Prior work has used in-context prompting of LLMs utilizes prompts to guide language models generating contextually relevant responses, which is generally more generalizable than fine-tuning. Some notable works here include Chain-of-thought (Wei et al., 2022), Zero-shot CoT (Kojima et al., 2022), among others. These, however, have no access or use external tools.

ART (Paranjape et al., 2023), and other concurrent studies (Lu et al., 2023; Qian et al., 2023) support accessing new tools through code or assembling tool sequences to generate the final response. Nonetheless, their way of accessing tools relies on extra task-specific information like demonstrations of how a task needs to be divided or conveyed to existing tools. This restricts their generalizability to new tasks that may necessitate new tools or a different combination of tools. Concurrent work (Hsieh et al., 2023) addresses this issue via documental tool descriptions. However, GEAR complements this work in that, our approach also uses tool outputs for more accurate tool grounding.

Another core issue in all these works is the tool grounding mechanism. Lu et al. (2023); Qian et al. (2023) rely solely on LLM prompting for tool grounding while ART applies cosine similarity query/tool representations for task grounding. However, little is understood about tradeoffs or limits of these approaches, which we explore in our experiments. To address these, our method extends these works and captures both semantic and pattern relationships (introduced in §3.1 and §3.2) between query and tools. This allows GEAR to successfully identify and utilize unseen tools for low-resource tasks (novel tasks) without the need for additional task information. Table 1 compares GEAR, CoT, Zero-shot CoT, Toolformer, and ART.

Embodied Language Model in Robotics. Recent research has focused on employing language models for robotic agents planning and their communication with the world (Driess et al., 2023; Zhao et al., 2023; Song et al., 2022; Huang et al., 2023; Vemprala et al., 2023). This is similar to the setup here involving a language model's interaction with external tools. Huang et al. (2022) and Lynch et al. (2022) leverage various sources of human language and textual feedback to guide robots while solving complex tasks. GEAR shares the same underlying idea with SayCan (Ahn et al., 2022) which utilizes binary scores for robotic affordance, while GEAR employs a distinct method that is designed for more general tool and task settings.

3 GEAR: Generalizable and Efficient Augmented Tool Resolution

We start with the formal problem statement. We are given an input query Q that we aim to solve. In addition, we are provided with a tool library $\mathcal{T} \triangleq \{(T_1, d_1, \pi_1), (T_2, d_2, \pi_2), \cdots, (T_n, d_n, \pi_n)\}$ with n tools. Each tool T_i can receive an API call (e.g., a question or a formula) and respond accordingly, often in the form of natural language. If the provided input is unparsable to the tool, it would return an empty response. Each tool is also supplied with its natural language description (d_i) and demonstrations (π_i) showing examples of natural language questions parsed by each tool.

GEAR aims to find the most appropriate tool for solving Q. As it can be observed in the Algorithm 1, GEAR iterates over the tools (line 2) and scores each tool i with respect to the given question Q (line 5). This score is a linear combination of two scores, a *semantic* similarity score $S(.,.)$ and a *pattern* similarity score $P(.,.)$. Semantic score (defined in §3.1) provides a measure of semantic alignment between the tool description d_i and the given query Q. Pattern similarity score (defined in §3.2) scores the alignment between the responses obtained from SLM and each tool, which provides an indication of how closely the tool's output aligns with a preliminary answer. The algorithm ultimately picks the most appropriate tool based on their scores (line 7) and obtains the final tool response via an API call generated by a LLM (line8, line9).

3.1 Semantic Similarity Score

Semantic similarity measures the alignment between the provided question to the language description of a tool. For instance, in Figure 2, the description of `Calculator` is semantically closer to a query that contains numbers, leading to a higher semantic score. Formally, this score is defined as:

$$S(Q, d_i) = f_{\text{SLM}}(Q, d_i),$$

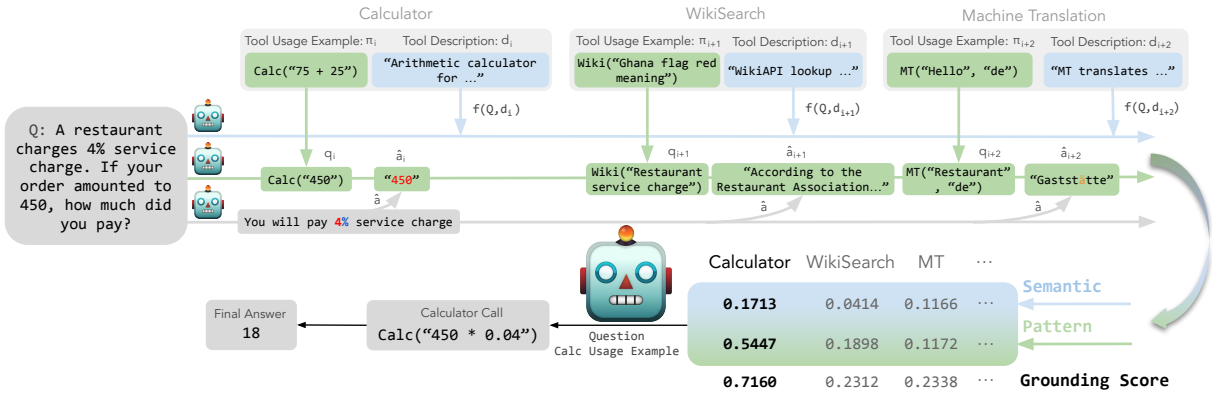


Figure 2: GEAR framework. It computes the pattern score by comparing the preliminary answer (in gray line) to tool responses (in green box) and the semantic score by comparing the query to tool descriptions (in blue box). Grounding tool with the highest weighted average score and executing it via a LLM to obtain the final answer.

Algorithm 1 GEAR Algorithm

Input: Query Q, Tool library $\mathcal{T}$, Small Language Model (SLM), Large Language Models (LLM)
Output: Grounded tool, and answer to the input question

1: $\hat{a} \xleftarrow{\text{sample}} \text{SLM}(Q)$
2: **for** (T_i, d_i, π_i) in $\mathcal{T}$ **do**
3: $\quad q_i \xleftarrow{\text{sample}} \text{SLM}(\pi_i + Q)$ ▷ Generate API call
4: $\quad \hat{a}_i \leftarrow T_i(q_i)$ ▷ Get the tool's response
5: $\quad f_i(Q) \leftarrow \gamma S(Q, d_i) + (1-\gamma) P(\hat{a}, \hat{a}_i)$ ▷ Score it
6: **end for**
7: $\iota \leftarrow \arg\max_i f_i(Q)$ ▷ Select the best tool
8: $q_\iota \xleftarrow{\text{sample}} \text{LLM}(\pi_\iota + Q)$ ▷ Generate API call
9: $a_\iota \leftarrow T_\iota(q_\iota)$ ▷ API call to the selected tool
10: **Return** grounded tool T_ι and the final answer a_ι.

where f is a similarity function utilizing the representation of SLM, quantifying the degree to which the query Q is semantically close to the tool description d_i. A popular choice to implement this similarity function (used in our experiments) is cosine distance between the representations query Q and tool description d_i:

$$S(Q, d_i) = \cos\left(\text{enc}_{\text{SLM}}(Q), \text{enc}_{\text{SLM}}(d_i)\right),$$

where $\text{enc}_{\text{SLM}}(.)$ is the representation of SLM.

3.2 Pattern Similarity Score

Pattern similarity provides an answer-level alignment score. This score computes an alignment between a preliminary guess $\hat{a}$ and the response generated by each tool $\hat{a}_i$. For instance, in Figure 2, the preliminary answer is "4", which has a higher pattern similarity score with `Calculator`'s response ("450", denoted in red), as both are numbers. Whereas, the responses from `Wiki` and `MT` are descriptive responses with a large proportion of English tokens (in black) and a non-ASCII token (in orange) that is not exhibited in the preliminary answer. Pattern similarity is computed based on the following steps.

Preliminary guess. First, SLM generates a zero-shot preliminary answer $\hat{a}$ for the given query using greedy decoding (line 1).[2]

Tool-based response. Then SLM is prompted by the given query and few shot usage examples to obtain API call q_i:

$$q_i \xleftarrow{\text{sample}} \text{SLM}(\pi_i + Q).$$

We then obtain the tool response $\hat{a}_i \leftarrow T_i(q_i)$ if q_i is parsable by the tool T_i, otherwise empty.

Scoring the alignment. The scoring is based on a predefined pattern set $\mathcal{S}$ consisting of distinct elements that correspond to output patterns of various tools. These pattern elements, for example, can represent numbers, English words, symbols, URLs, or certain robotic movements.[3] We encode raw tool response $\hat{a}_i$ to its corresponding pattern set $\{e_j(t) \mid \forall j \in \{1, 2, \cdots, |\mathcal{S}|\}, \forall t \in \hat{a}_i\}$, where t is the word token of $\hat{a}_i$ and the encoding function $e_j : t \rightarrow \mathcal{S}$ encodes word token to the j^{th} pattern of $\mathcal{S}$ if token exhibits that pattern, otherwise empty.[4] Formally, the output of e_j for t is ei-

[2] We recommend greedy decoding for this zero-shot SLM-based step to reduce the risk of significantly poor responses which may occur in stochastic decoding.

[3] While our evaluation is focused on language tools, the idea discussed here should in principle generalize to other modalities such as physical tools.

[4] For instance, if $\mathcal{S}$ = `{e,f,n}` consisting of English, non-ASCII and number patterns respectively, the sentence "Hello World 2023" would be encoded to `{e,e,n}`. If multiple patterns are exhibited in one word token, each pattern would be encoded separately: the German word "`lächeln`" $\Longrightarrow$`{e,f,e}`.

ther a multiset of j^{th} pattern ($\{\mathcal{S}_j^1, \cdots, \mathcal{S}_j^n\}$ where $n \geq 1$) or an empty set ϕ. Thus, the final encoded pattern set of $\hat{a}_i$ is the multisubset of $\mathcal{S}$. The encoding of $\hat{a}$ follows the same procedure. Let $C_j^{\hat{a}}$ and $C_j^{\hat{a}_i}$ denote the number of j^{th} pattern encoded by e_j in the pattern set of $\hat{a}$ and $\hat{a}_i$. Namely, for $\hat{a}_i$, $C_j^{\hat{a}_i} = |\{e_j(t) \mid \forall t \in \hat{a}_i\}|$. Let $|\hat{a}|$ and $|\hat{a}_i|$ be the length of final encoded pattern sets of $\hat{a}$ and $\hat{a}_i$. The pattern similarity score between tool response $\hat{a}_i$ and preliminary answer $\hat{a}$ is computed as:

$$P(\hat{a}, \hat{a}_i) = \sum_{j \in \{1,\cdots,|\mathcal{S}|\}} \frac{(C_j^{\hat{a}} + \lambda) C_j^{\hat{a}_i}}{(|\hat{a}| + \lambda|\mathcal{S}|)|\hat{a}_i|} \log \frac{1}{\mathcal{P}_j},$$

where $\mathcal{P}_j$ is the prior probability of the j^{th} pattern from a prior pattern distribution $\mathcal{P}$. $\mathcal{P}, \mathcal{S}$ and e_j can be shared across different task and tool library settings. Add-λ smoothing is applied to solve the pattern zero-frequency issue. However, if $\hat{a}_i$ is empty, $P(\hat{a}, \hat{a}_i)$ will be assigned its lower bound value 0. In our experiment, we use regular expressions as encoding functions e_j.

Intuitively, the pattern similarity score $P(\hat{a}, \hat{a}_i)$ is the cross entropy between the prior pattern distribution $\mathcal{P}$ and the smoothed joint pattern distribution from true tool response $\hat{a}_i$ and preliminary answer $\hat{a}$. It is proved to have strict lower and upper bounds in Appendix A.1 and holds the following five essential properties: (i) *Order Insensitive* (ii) *Length Insensitive* (iii) *Pattern Sensitive* (iv) *Pattern Set Size Insensitive* (v) *Commutative*. Explanations and proofs of these properties are provided in Appendix A.2.

We hypothesize that tools could easily elicit their latent pattern distribution through parsable API calls, irrespective of its correctness. Therefore, despite their less reliable performance, SLMs are sufficient for query-tool grounding, because their key task is to generate appropriate response patterns in $\hat{a}$ for the given query and parsable API call q_i for the target tool, which is much simpler than reasoning to make $\hat{a}$ (zero-shot result without tool use) or q_i (API call for result with tool use) correct. In Appendix A.3, we discuss mock responses which can further enhance the efficiency and generalizability of the grounding process.

4 Experiment Setup

4.1 GEAR Implementation.

We implement GEAR according to the construction described in §3. Throughout the experiments the LLMs in our study are `GPT-J` and $\texttt{GPT3}_{\texttt{davinci-003}}$ (in short, `GPT-3`), and our SLMs are `GPT-Neo`, $\texttt{GPT2}_{\texttt{medium}}$, $\texttt{GPT2}_{\texttt{large}}$, `MiniLM` and `MPNet`.[5]

Specifically for our implementation of GEAR, we use `MPNet` to calculate semantic similarity scores and `GPT-Neo` for generating preliminary answers and API calls to calculate pattern similarity scores. For LLMs, we use either `GPT-J` or `GPT-3` for final tool execution.

Algorithm →	Zero-shot	Few-shot	$\text{ART}^*_{\text{llm}}$	GEAR
Grounding Model →	–	–	`GPT-Neo`	`GPT-Neo`
Execution Model →	`GPT-J`	`GPT-J`	`GPT-J`	`GPT-J`
Datasets ↓				
ASDiv	7.5	21.4	16.7	23.3
GSM8K	0.4	5.6	9.8	3.8
SVAMP	2.0	13.1	11.2	18.6
↳ **Average (Arithm)**	3.3	13.4	12.6	**15.2**
IWSLT (cn)	10.5	16.9	4.1	21.1
IWSLT (ar)	8.5	18.7	4.8	17.6
IWSLT (de)	7.7	19.3	5.4	32.9
IWSLT (fr)	7.9	22.7	6.7	38.4
IWSLT (ja)	5.5	14.4	3.4	12.9
IWSLT(ko)	8.9	15.2	3.6	14.9
↳ **Average (MT)**	8.2	17.9	4.7	**23.0**
NQ-Open	10.2	31.1	21.2	43.4
WebQS	5.3	18.2	11.2	22.1
TriviaQA	27.3	46.5	29.3	50.3
↳ **Average (ODQA)**	14.3	31.9	20.6	**38.6**
CSQA	10.9	37.1	6.3	60.7
COPA	6.5	27.0	1.0	13.6
SocialIQA	8.4	26.0	5.5	41.5
↳ **Average (CSQA)**	8.6	30.0	4.3	**38.6**

Table 2: Downstream task performance results (§5.1). Evidently, GEAR**-augmented** `GPT-J` **outperforms our baselines** when using a consistent set of grounding and execution models.

Tools. To evaluate the performance for a variety of purposes, we create a total of 10 different tools, including 4 basic tools: `Calculator`, `MT`, `Wiki`, and `QA`; and 6 novel tools: `Timezone Converter`, `Multilingual QA`, `Sleep`, `Exponential Calculator`, `Logarithmic Calculator`, and `Movement Controller`. All of them are accessible via specific API calls and have corresponding returns. Examples of API calls are shown in Table 13 and more information about tools can be found in Appendix C.

Datasets. We conduct our experiment on 14 datasets across 6 downstream tasks. The dataset

[5]We accessed the OpenAI models on April through June, 2023.

Models	ASDiv	SVAMP	SQuAD	T-REX	TriviaQA	MLQA(es)
Toolformer (GPT-J)	40.4	29.4	33.8	53.5	48.8	20.6
ART$^{*}_{\text{llm}}$ (GPT-Neo/GPT-3)	37.0	21.3	17.7	20.6	24.3	14.0
ART$_{\text{cs}}$ (MiniLM/GPT3$_{\text{davinci-003}}$)	**86.7**	77.3	39.3	50.4	61.0	–
GEAR (GPT-Neo/GPT3$_{\text{davinci-003}}$)	74.9 (-11.8)	**79.9 (+2.6)**	**61.1 (+21.8)**	**83.1 (+32.7)**	**62.5 (+1.5)**	**58.3 (+37.7)**

Table 3: Comparing GEAR with Toolformer (Schick et al., 2023) and ART (Paranjape et al., 2023) (§5.1). The original ART work, ART$_{\text{cs}}$, employs MiniLM for cosine similarity strategy and does not have QA or MT for the MLQA task.

Models	Evaluate on → Demonstration ↓	ASDiv	GSM8K	SVAMP	TriviaQA	NQ-Open	WebQS
ART$_{\text{cs}}$ (MiniLM/GPT3$_{\text{davinci-003}}$)	ASDiv	97.9	88.5	87.2	2.1	1.4	0.0
	GSM8K	93.8	88.4	81.9	0.3	1.1	0.0
	SVAMP	98.3	74.5	75.7	0.0	1.1	0.0
	TriviaQA	25.8	32.2	22.5	98.1	96.2	0.4
	NQ-Open	25.3	25.2	22.4	97.4	98.2	0.4
	WebQS	28.6	39.9	28.3	94.8	96.8	1.1
GEAR (GPT-Neo/GPT3$_{\text{davinci-003}}$)		83.1	83.0	89.0	63.0	65.6	54.3

Table 4: Cross-dataset generalization evaluation of tool grounding accuracy (§5.2). Evidently, GEAR **can identify the appropriate tool for a given task without requiring in-domain demonstrations** while ART has a significant grounding performance decline on out-domain demonstrations, with each score representing grounding accuracy/affordance ratio in percentage.

details and evaluation metrics can be found in Appendix B.4.

4.2 Baseline Systems

We organize our baselines as follows:

- **Zero-shot**: This baseline directly asks questions to LLM without any instruction.
- **Few-shot**: This baseline involves prompting LLM with natural language instructions that articulate the requirements of the given task.
- **ART**: This approach uses prompting LLM for multi-step reasoning and tools execution (Paranjape et al., 2023). Besides the results in the original paper, we experiment with a reimplementation of ART (referred to as ART*) adapted to our tools and tasks. Specifically, following the original work, we report two variants of this model with different tool-grounding strategies proposed in its paper: (1) LLM-based prompting similarity (ART$^{*}_{\text{llm}}$) and (2) cosine similarity (ART$^{*}_{\text{cs}}$).

To ensure a fair comparison between baselines, we let few-shot, ART*, and GEAR use the same prompt examples (Appendix H).

5 Experimental Findings

We compare the downstream performances of models (§5.1), and compare their generalizability to new tools or tasks (§5.2).

5.1 Results on Downstream Tasks

We first evaluate all our models on the downstream task performance with a tool library containing 4 basic tools (Table 2). For consistency of comparisons, all the baselines use GPT-J for the final answer execution. GEAR outperforms all the baselines across four basic tasks. For example, the accuracy of GEAR-augmented GPT-J is 24.3% and 6.7% higher than zero-shot and few-shot baselines on the ODQA (Open-domain QA) task. Compared to the ART$^{*}_{\text{llm}}$, GEAR consistently has superior performance because of better tool use. Later in §5.2 we show that this performance gap is due to the difference in tool grounding accuracy. Additional results using GPT-3 as execution model (in place of GPT-J) are provided in Appendix D.

Table 3 puts Toolformer (Schick et al., 2023), ART (Paranjape et al., 2023) and GEAR together, evaluating on their shared datasets. All datasets are evaluated under a 4 basic tools library except for MLQA which uses a 5-tools library with an extra Multilingual QA tool. Since Toolformer code and model are not available online, we are not able to reproduce their results and therefore, copy the numbers from its paper. The comparison is unfair to Toolformer as it uses a finetuned GPT-J model. But it is informative that GEAR-

Algorithm →		GEAR			ART^{*}_{llm}	ART^{*}_{llm}	ART^{*}_{cs}
Grounding Model → **Dataset (w/ 4 Tools) ↓**	**Target Tool ↓**	GPT-Neo (1.3B)	$GPT2_{large}$ (774M)	$GPT2_{medium}$ (355M)	GPT-Neo (1.3B)	$GPT3_{davinci-003}$ (175B)	MPNet (110M)
ASDiv	Cal	83.1	77.7	58.7	25.6	46.5	98.8
GSM8K	Cal	83.0	65.3	55.6	38.0	45.5	99.5
SVAMP	Cal	89.0	76.5	65.1	21.0	50.0	100.0
↳ **Average (Arithm)**		**85.0**	73.2	59.8	28.2	47.3	**99.4**
IWSLT (cn)	MT	84.1	95.5	98.2	30.0	63.2	99.9
IWSLT (ar)	MT	66.6	–	–	27.8	61.6	98.6
IWSLT (de)	MT	96.9	94.4	95.2	31.6	66.0	94.0
IWSLT (fr)	MT	96.6	94.0	96.0	33.8	64.4	92.2
IWSLT (ja)	MT	72.4	89.3	91.1	30.8	62.8	97.8
IWSLT (ko)	MT	82.2	66.7	91.7	25.9	72.7	99.4
↳ **Average (MT)**		83.1	88.0	**94.4**	30.0	65.1	**97.0**
NQ-Open	Wiki	63.0	61.3	59.1	10.9	44.0	39.4
WebQS	Wiki	65.6	83.1	81.4	13.6	56.8	60.5
TriviaQA	Wiki	54.3	77.2	71.7	13.2	58.1	41.5
↳ **Average (ODQA)**		61.0	**73.9**	70.7	12.6	**53.0**	47.1
CommonsenseQA	QA	77.1	84.0	84.9	10.1	34.9	69.7
COPA	QA	41.3	77.2	61.2	7.2	24.4	29.7
SocialIQA	QA	75.7	87.6	59.5	16.4	42.4	14.1
↳ **Average (CSQA)**		64.7	**82.9**	68.5	11.2	**33.9**	37.8
# of Operation in GFLOPS[6]		1573	937	430	5455	728420[6]	160

Table 5: Tool grounding accuracy for 4 downstream tasks with a 4-tools library (§5.2). **Bold** denotes the highest value within its grounding strategy and **underline** represents the highest among all baselines. We find that GEAR **yields better performance compared to the LLM-based strategy on all datasets.** GEAR **is generalizable to smaller SLMs and even achieve better grounding results on certain tasks**.

augmented GPT-3 outperforms the original work ART_{cs}, which employs the same-sized model with task-specific demonstrations, on 4 out of 5 tasks. This performance gain also emphasizes the strong generalization capability of GEAR.

5.2 Results on Tool Grounding

We systematically examine the tool grounding accuracy (the percentage of correctly selected tools) across a variety of tool library sizes and model sizes. We first calculate the grounding accuracy for a tool library comprising 4 basic tools. Then we expand the tool library to a total of 10, as described in Appendix C.2, by introducing competitor and distractor tools. We re-evaluate the grounding accuracy for the four basic tasks, along with two novel tasks requiring `Multilingual QA` and `Timezone Converter` tools. The main results are shown in Table 5 and Figure 3.

GEAR **is more generalizable than other query-tool grounding algorithms.** According to Table 5, GEAR utilizing GPT-Neo with 1.3B parameters significantly outperforms the LLM-based strategy proposed by ART (Paranjape et al., 2023), even when the latter uses GPT-3 which is 134 × larger. The best-reported similarity strategy in ART, which calculates the cosine similarity between the given demonstration and textual description of tasks, performs outstandingly well on Arithmetic and MT tasks. We hypothesize this is because of the presence of distinct and unique keywords in Arithmetic and MT queries, which are easily distinguishable by word embeddings. However, for more open-ended NLP tasks like Open-domain and Commonsense QA, word embeddings are less generalizable in selecting the correct tools, resulting in low grounding accuracy of 47.1% and 37.8%. In contrast, GEAR's grounding strategy is shown to be more strong with grounding accuracy of 61.0% and 64.7% on the aforementioned tasks.

Table 4 displays a substantial decline in grounding accuracy of ART (Paranjape et al., 2023) when using out-domain demonstrations. In contrast, GEAR consistently maintains its high performance without requiring in-domain demonstrations. We also demonstrate GEAR outperforms retrieval-

[6]Since OpenAI has not open sourced their $GPT3_{davinci-003}$, we approximate the operations as # tokens × # params, which is the lower bound of operations. The real amount of operations should exceed this estimation.

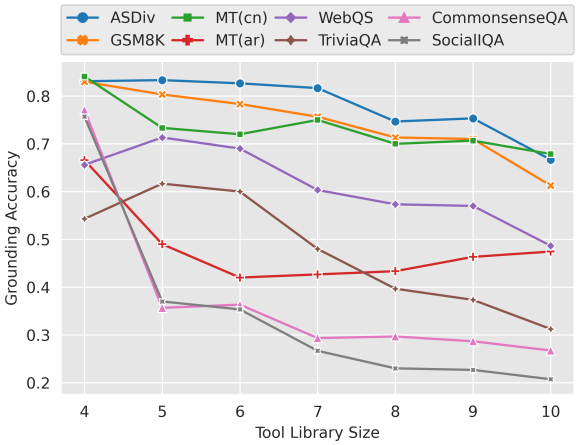


Figure 3: Grounding accuracy of GEAR when the tool library is expanded from 4 to 10 tools (§5.2). We incrementally incorporate these tools: `Multilingual QA`, `Timezone Converter`, `Sleep`, `Logarithmic Calculator`, and `Movement Controller`.

based baselines on query-tool grounding, as shown in Table 11 in Appendix E.

GEAR **is generalizable to smaller language models.** We evaluate the grounding performance of GEAR on two smaller GPT-2 models. As reported in Table 5, GEAR consistently exhibits high-level grounding accuracy on both SLMs and even outperforms `GPT-Neo` on certain tasks. For example, GEAR-augmented $\texttt{GPT2}_{\texttt{large}}$ achieves 73.9% and 82.9% grounding accuracy for the Open-domain QA and Commensense QA tasks, greatly higher than those of ART* baselines. Moreover, as the model size increases, the marginal grounding accuracy gain diminishes. This is because as long as the SLM produces expected patterns for the given query, the correctness of the preliminary answer has no bearing on the pattern similarity score (see case study in §6.2). Which, in turn, experimentally proves the feasibility of employing SLMs for query-tool grounding.

GEAR **is generalizable to larger tool libraries.** Because of a more comprehensive grounding process, GEAR enables certain tasks to generalize better for larger sets of tools. Figure 3 displays the grounding accuracy changing from 4 to 10 tools. The general low decreasing rates for Arithmetic, MT and Open-domain QA demonstrate the ability of GEAR in handling tool libraries of varying sizes.

We hypothesize the drops between the fourth and fifth tools of CommonsenseQA and SoicalIQA datasets are likely due to the introduction of the `Multilingual QA` tool which has functional overlap with the basic `QA` tool. Specifically, the `Multilingual QA` tool can also solve reasoning tasks by translating contexts from English to English; therefore, if we consider `Multilingual QA` as the correct tool for the Commonsense QA task as well, the averaged final grounding accuracy of Commonsense QA task will increase to 49.1%, with a 15.6% decrease compared to Table 5.

We also compare GEAR and the best variant ART^*_{cs} under a 10-tools library on 6 downstream tasks with two extra novel tasks. In short, GEAR outperforms ART^*_{cs} on 5 out of 6 tasks. See Appendix E for detailed results.

Task	GEAR	Performance change Δ ¬Pattern Sim	¬Semantic Sim
Arithm	74.0	**-2.3**	**-11.5**
MT	80.5	+10.9	-69.9
ODQA	40.7	**-15.4**	**-21.1**
CSQA	33.4	**-21.6**	**-18.9**
MLQA	54.4	**-10.6**	**-31.5**
TZ Conversion	96.4	+3.6	-94.9

Table 6: The result of leave-one-out ablation study for 10-tools library (§6.1). **The decrease in grounding accuracy on both columns demonstrates the importance of considering both semantic and pattern scores for query-tool grounding.**

6 Analysis

6.1 Ablation Study

We now perform a leave-one-out experiment to better understand the contribution of each score (§3.1 and §3.2) to the final grounding accuracy. We conduct experiments for a 10-tools library with only either semantic similarity score or pattern similarity score. The results are shown in Table 6. For the 10-tools library, there are 4 out of 6 tasks displaying grounding accuracy decline in both semantic and pattern columns, suggesting that it is crucial to consider both semantic and pattern similarity scores for query-tool grounding. Tasks such as MT and Timezone Conversion show increased grounding accuracy in the semantic column, which is likely due to the same reason discussed in §5.2: these two tasks contain unique keywords so that single semantic similarity score suffices to distinguish them from other tasks (more results in Appendix F.)

6.2 Case Study on SLM's Size

It is natural to question whether GEAR will have much better performance if we replace SLM with

[*Question*] Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
[*Preliminary answer from* GPT2$_{\text{medium}}$] i make $ 2
[*Preliminary answer from* GPT2$_{\text{large}}$] the average american spends $ 1000 a year on food
[*Preliminary answer from* GPT-Neo] janet's ducks lay 16 eggs per day
[*Preliminary answer from* GPT-3] Janet eats 3 of the 16 eggs, so she has 16 - 3 = «16-3=13»13 eggs left. She bakes muffins with 4 of these eggs, so she has 13 - 4 = «13-4=9»9 eggs left to sell. Thus, she makes $ 9 * 2 = $ «9*2=18»18 every day at the farmers' market. Answer: $ 18.

[*Question*] In which decade did Billboard magazine first publish and American hit chart?
[*Preliminary answer from* GPT2$_{\text{medium}}$] the first billboard magazine was published in the fall of 1892.
[*Preliminary answer from* GPT2$_{\text{large}}$] the first billboard magazine was published in the united states in the year 900.
[*Preliminary answer from* GPT-Neo] the billboard hot 100 is a weekly chart that ranks the most popular songs in the United States
[*Preliminary answer from* GPT-3] The 1930s. Specifically, Billboard magazine first published its "Music Popularity Chart" in 1936.

Figure 4: A comparison of output patterns between SLMs and LLM. The lines subsequent to [*Question*] represents the output generated by the corresponding model, with patterns (number, symbol and English alphabet) labeled in different colors. While SLMs tend to be less accurate than LLM, **their responses provide sufficient clues (pattern distribution) about the form of the expected answer**.

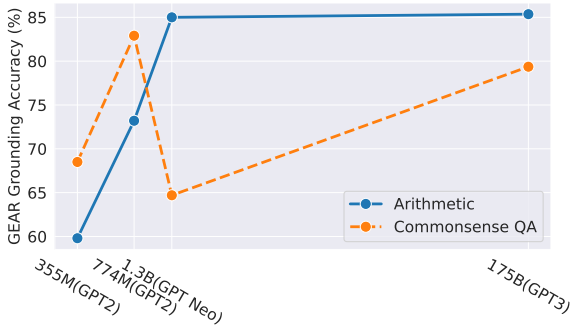


Figure 5: Averaged GEAR grounding performance over SLM sizes (number of parameters, in log scale) on Arithmetic and Commonsense QA tasks. Each task is evaluated by three datasets. GEAR **with SLM has a similar grounding accuracy as with LLM**.

LLM, namely, $\hat{a} \xleftarrow{\text{sample}} \text{LLM}(Q)$ in Algorithm 1. We provide a case study (Figure 4) showing the impact of various SLM choices, including the setting where replacing SLM with LLM, to further illustrate our observation in §5.2 that as the model size increases, the marginal grounding accuracy gain diminishes (Figure 5). In the first example from GSM8K (Cobbe et al., 2021), we can see that SLM offers the similar indicative signal as LLM that the potential answer should contain number and symbol patterns, despite their responses being incorrect. We also observe that this phenomenon not only happens in pattern-specific tasks (e.g. Arithmetic) but also occurs in more general open-ended tasks like Commonsense QA. The second TriviaQA (Joshi et al., 2017) example shows that the pattern distributions generated by the SLMs closely resemble the LLM's distribution: a single number amid English text.

Thus as long as API calls are properly generated, it is highly likely that GEAR with SLM will select the same tool as with LLM. In other words, generating executable API calls from SLM now becomes the only empirical limitation of the upper bound of the pattern similarity score. As the model size increases, this limitation will become less strict, resulting in a diminished rate of improvement in grounding performance.

To validate the above observations, we visualize the grounding performance of GEAR across different SLM sizes on these two tasks in Figure 5. Evidently, as the increasing of SLM sizes, the grounding performance margin tends to decrease. Note that because of different model families, SLM grounding performance may not necessarily be monotonically increasing (orange line).

7 Conclusion

In this paper, we introduce GEAR: a generalizable query-tool grounding algorithm that enables efficient tool groundings without extra fine-tuning or task-specific demonstrations. This is accomplished by introducing a fine-grained scoring mechanism that leverages both semantic and pattern similarities and leveraging smaller language models for query-tool grounding. To validate the generalizability of GEAR, we conduct extensive experiments that demonstrate its capacity to deal with large tool libraries and novel tasks.

Limitations

While GEAR aims to improve the query-tool grounding and exhibits strong generalization and robustness for large tool libraries, including user-provided pipelines, it has a potential limitation in lacking support for automatic tool pipeline construction. Future works could focus on how to combine GEAR with automatic reasoning and task decomposition works, such as ART (Paranjape et al., 2023), Chameleon (Lu et al., 2023), and CREATOR (Qian et al., 2023). We believe that the combination of generalizable and efficient tool grounding with multi-hop reasoning would further boost the performance of the current SOTA LLMs.

Theoretically, GEAR supports tools that have non-textual returns via mock responses. However, we only test the `Sleep` and `Movement Controller` tools in the main experiment and the `Image Generation` tool in the GEAR-augmented chatbot. Though achieving promising results on these three tools, future works, especially in the embodied LM area, could further explore how mock responses can be used in grounding human language with physical world tools.

Acknowledgements

The authors would like to thank Adam Byerly, Tianjian Li, and Zhengping Jiang for their helpful input on earlier versions of this work. The authors would like to acknowledge the generous gifts from Amazon and the Allen Institute for AI. Moreover, the authors would like to thank the Center for Language and Speech Processing members at Johns Hopkins University for their valuable feedback and comments. GPU machines for conducting experiments were provided by ARCH Rockfish cluster (`https://www.arch.jhu.edu`).

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Conference on Empirical Methods in Natural Language Processing* (EMNLP).

Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Jan Niehues, Sebastian Stüker, Katsuhito Sudoh, Koichiro Yoshino, and Christian Federmann. 2017. Overview of the IWSLT 2017 evaluation campaign. In *Proceedings of the 14th International Conference on Spoken Language Translation*, pages 2–14, Tokyo, Japan. International Workshop on Spoken Language Translation.

Zhipeng 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*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

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*.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. 2018. T-REx: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Evelyn Fix and Joseph Lawson Hodges. 1989. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

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*.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. 2023. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*.

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

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2021. Text modular networks: Learning to decompose tasks in the language of existing models. In *Conference of the North American Chapter of the Association for Computational Linguistics* (NAACL).

Tushar Khot, Kyle Richardson, Daniel Khashabi, and Ashish Sabharwal. 2022. Hey ai, can you solve complex tasks by talking to agents? In *Annual Meeting of the Association for Computational Linguistics* (ACL) - *Findings*.

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* (NeurIPS).

Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2022. Internet-augmented dialogue generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8460–8478, Dublin, Ireland. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Annual Meeting of the Association for Computational Linguistics* (ACL), pages 6086–6096.

Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2020. MLQA: Evaluating cross-lingual extractive question answering. In *Annual Meeting of the Association for Computational Linguistics* (ACL), pages 7315–7330.

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*.

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*.

Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. 2022. Interactive language: Talking to robots in real time. *arXiv preprint arXiv:2210.06407*.

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*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Annual Meeting of the Association for Computational Linguistics* (ACL), pages 975–984, Online. 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*.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Conference of the North American Chapter of the Association for Computational Linguistics* (NAACL), pages 2080–2094, Online. Association for Computational Linguistics.

Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Disentangling abstract and concrete reasonings of large language models through tool creation. *arXiv preprint arXiv:2305.14318*.

Shuofei Qiao, Honghao Gui, Huajun Chen, and Ningyu Zhang. 2023. Making language models better tool learners with execution feedback. *arXiv preprint arXiv:2305.13068*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing* (EMNLP).

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

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*.

Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. 2022. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2022. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Conference of the North American Chapter of the Association for Computational Linguistics* (NAACL), pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. LaMDA: Language Models for Dialog Applications. *arXiv preprint arXiv:2201.08239*.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

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*.

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

Xufeng Zhao, Mengdi Li, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter. 2023. Chat with the environment: Interactive multimodal perception using large language models. *arXiv preprint arXiv:2303.08268*.

Supplemental Material

A Pattern Similarity Score

A.1 Pattern Similarity Score Bounds

Because the count C and λ are nonnegative, $\mathcal{P}_j \in [0, 1]$, $|\hat{a}_i|$ and $|\hat{a}|$ indicate the total number of encoded patterns from tool response and preliminary answer, we always have $P(\hat{a}, \hat{a}_i) \geq 0$. In this proof, for better understanding, we assume a most common case that each word token is encoded to only one pattern, namely no word token exhibits multiple patterns. Thus, $|\hat{a}_i|$ and $|\hat{a}|$ are equal to the length of unencoded sequences of tool response and preliminary answer. $p_{\hat{a}_i}(\cdot) = \frac{C_j^{\hat{a}_i}}{|\hat{a}_i|}$ and $p_{\hat{a}}(\cdot) = \frac{C_j^{\hat{a}}}{|\hat{a}|}$ represent the probability of j^{th} pattern in raw $\hat{a}_i$ and $\hat{a}$.

$$
\begin{aligned}
P(\hat{a}, \hat{a}_i) &= \sum_{j \in \{1, \cdots, |\mathcal{S}|\}} \frac{(C_j^{\hat{a}} + \lambda) C_j^{\hat{a}_i}}{(|\hat{a}| + \lambda|\mathcal{S}|)|\hat{a}_i|} \log \frac{1}{\mathcal{P}_j} \\
&= \sum_{j \in \{1, \cdots, |\mathcal{S}|\}} \frac{C_j^{\hat{a}} C_j^{\hat{a}_i} + \lambda C_j^{\hat{a}_i}}{|\hat{a}||\hat{a}_i| + \lambda|\mathcal{S}||\hat{a}_i|} \log \frac{1}{\mathcal{P}_j}
\end{aligned}
$$

If $\lambda = 0$:

$$
\begin{aligned}
& P(\hat{a}, \hat{a}_i) \\
&= \sum_{j \in \{1, \cdots, |\mathcal{S}|\}} \frac{C_j^{\hat{a}} C_j^{\hat{a}_i}}{|\hat{a}||\hat{a}_i|} \log \frac{1}{\mathcal{P}_j} \\
&= \sum_{x \in \{E(\hat{a}_i)\}} \sum_{y \in \{E(\hat{a})\}} p_{\hat{a}_i, \hat{a}}(x, y) \mathbb{I}_{x=y} \log \frac{1}{\mathcal{P}(x)} \\
&= \sum_{x \in \{E(\hat{a}_i)\}} \sum_{y \in \{E(\hat{a})\}} p_{\hat{a}_i}(x) p_{\hat{a}}(y) \mathbb{I}_{x=y} \log \frac{1}{\mathcal{P}(x)} \\
&= \sum_{x \in \{E(\hat{a}_i)\}} p_{\hat{a}_i}(x) \log \frac{1}{\mathcal{P}(x)} \sum_{y \in \{E(\hat{a})\}} p_{\hat{a}}(y) \mathbb{I}_{x=y} \\
&\leq \sum_{x \in \{E(\hat{a}_i)\}} p_{\hat{a}_i}(x) \log \frac{1}{\mathcal{P}(x)} \cdot p_{\hat{a}}(x) \\
&\quad \text{(Because it is possible } \{E(\hat{a}_i)\} \cap \{E(\hat{a})\} = \phi) \\
&= \sum_{x \in \{E(\hat{a}_i)\}} p_{\hat{a}_i}(x) p_{\hat{a}}(x) \log \frac{1}{\mathcal{P}(x)} \\
&\leq \sum_{x \in \{E(\hat{a}_i)\}} p_{\hat{a}_i}(x) \log \frac{1}{\mathcal{P}(x)} \\
&= \text{CE}(p_{\hat{a}_i}, \mathcal{P})
\end{aligned}
$$

Properties	$\hat{a}$ (Encoded)	$\hat{a}_1$ (Encoded)	$\hat{a}_2$ (Encoded)	**Result**
Order Insensitive	`ene`	`ene`	`een`	$P(\hat{a}, \hat{a}_1) = P(\hat{a}, \hat{a}_2)$
Length Insensitive	`eee`	`en`	`enenen`	$P(\hat{a}, \hat{a}_1) = P(\hat{a}, \hat{a}_2)$
Pattern Sensitive	`ene`	`ene`	`enn`	$P(\hat{a}, \hat{a}_1) < P(\hat{a}, \hat{a}_2)$
Commutative	`ene`	`eee`	`nnn`	$P(\hat{a}, \hat{a}_1) = P(\hat{a}_1, \hat{a})$

Table 7: Examples illustrating the four essential properties of pattern similarity scores

$\text{CE}(p_{\hat{a}_i}, \mathcal{P})$ is the cross-entropy between the pattern distribution of raw tool response and the prior pattern distribution. $\{E(\hat{a}_i)\}$ and $\{E(\hat{a})\}$ are two sets of patterns derived from encoding tool response $\hat{a}_i$ and preliminary answer $\hat{a}$, respectively. $p_{\hat{a}_i,\hat{a}}(x, y)$ is the joint probability of pattern x and y in $\hat{a}_i$ and $\hat{a}$. Because $\hat{a}_i$ and $\hat{a}$ are obtained independently, we can simply write the joint probability as the product of $p_{\hat{a}_i}(x)$ and $p_{\hat{a}}(y)$. $\mathbb{I}_{x=y}$ is the indicator function. $\mathcal{P}(x)$ is the prior probaility of the pattern x. Note that unlike j which is an index variable, x and y here are real pattern variables.

If $\lambda > 0$: let $\delta \subseteq \{1, 2, \cdots, |\mathcal{S}|\}$ such that $C_\alpha^{\hat{a}} > 0$ for $\alpha \in \delta$ and $C_\beta^{\hat{a}} = 0$ for $\beta \in \{1, 2, \cdots, |\mathcal{S}|\} \setminus \delta$.

$$
\begin{aligned}
& P(\hat{a}, \hat{a}_i) \\
&= \sum_{\alpha \in \delta} \frac{C_\alpha^{\hat{a}} C_\alpha^{\hat{a}_i} + \lambda C_\alpha^{\hat{a}_i}}{|\hat{a}||\hat{a}_i| + \lambda|\mathcal{S}||\hat{a}_i|} \log \frac{1}{\mathcal{P}_\alpha} \\
&\qquad + \sum_{\beta \in \{1,2,\cdots,|\mathcal{S}|\} \setminus \delta} \frac{\lambda C_\beta^{\hat{a}_i}}{|\hat{a}||\hat{a}_i| + \lambda|\mathcal{S}||\hat{a}_i|} \log \frac{1}{\mathcal{P}_\beta} \\
&= \sum_{\alpha \in \delta} \frac{C_\alpha^{\hat{a}_i}}{|\hat{a}_i|} \cdot \frac{C_\alpha^{\hat{a}} + \lambda}{|\hat{a}| + \lambda|\mathcal{S}|} \log \frac{1}{\mathcal{P}_\alpha} \\
&\qquad + \sum_{\beta \in \{1,2,\cdots,|\mathcal{S}|\} \setminus \delta} \frac{C_\beta^{\hat{a}_i}}{|\hat{a}_i|} \cdot \frac{\lambda}{|\hat{a}| + \lambda|\mathcal{S}|} \log \frac{1}{\mathcal{P}_\beta} \\
&\leq \sum_{\alpha \in \delta} \frac{C_\alpha^{\hat{a}} + \lambda}{|\hat{a}| + \lambda|\mathcal{S}|} \log \frac{1}{\mathcal{P}_\alpha} \\
&\qquad + \sum_{\beta \in \{1,2,\cdots,|\mathcal{S}|\} \setminus \delta} \frac{\lambda}{|\hat{a}| + \lambda|\mathcal{S}|} \log \frac{1}{\mathcal{P}_\beta} \\
&= \text{CE}_\alpha(\tilde{p}_{\hat{a}}, \mathcal{P}) + \lambda \text{CE}_\beta(U(0, |\hat{a}| + \lambda|\mathcal{S}|), \mathcal{P})
\end{aligned}
$$

where U is the uniform distribution and $\tilde{p}_{\hat{a}}$ is the smoothed pattern distribution of $\hat{a}$.

A.2 Pattern Similarity Score Properties

- *Order Insensitive:* The position of a pattern should not influence the score, as the preliminary answer generated by the SLM tends to be disorganized.
- *Length Insensitive:* The score should not be biased toward the length of tools' responses, as certain tools are inclined to generate longer responses.
- *Pattern Sensitive:* Given the prior distribution $\mathcal{P}$, tools that exhibit rare patterns are more likely to be chosen when the preliminary answer $\hat{a}$ also exhibits those patterns.
- *Pattern Set Size Insensitive:* The average pattern similarity score should remain consistent for various tool library and pattern set sizes. This property ensures a consistent hyperparameter γ (the weight for semantic and pattern scores).
- *Commutative:* $P(\hat{a}, \hat{a}_i) = P(\hat{a}_i, \hat{a})$ should be hold for any preliminary answer $\hat{a}$ and tool responses $\hat{a}_i$.

Table 7 gives illustrative examples for the pattern similarity score. `e` and `n` denote English token pattern

and number pattern. The less frequency of numbers "n" in real corpus compared to English tokens "e" results in a smaller prior probability $\mathcal{P}(n) < \mathcal{P}(e)$, leading to the result in the Pattern Sensitive row. In other words, with the same length, tool response $\hat{a}_2$ containing more rare patterns which also exhibit in the preliminary answer $\hat{a}$ would have higher pattern similarity score.

The Pattern Set Size Insensitive property also holds because the denominator $(|\hat{a}| + \lambda|\mathcal{S}|)|\hat{a}_i|$ is insensitive to the $\Delta|\mathcal{S}|$, given that $|\hat{a}| \gg |\mathcal{S}|$ and λ is typically small. Therefore, as long as the tool response or preliminary answer does not exhibit the given patterns, namely $C_j^{\hat{a}} = 0$ or $C_j^{\hat{a}_i} = 0$, $P(\hat{a}, \hat{a}_i)$ would not significantly change regardless of the size of $|\mathcal{S}|$.

To prove the length-insensitive property, we have to first assume tool responses $\hat{a}_1$ and $\hat{a}_2$ share the same pattern probability distribution. Namely, we have

$$\frac{C_j^{\hat{a}_1}}{|\hat{a}_1|} = \frac{C_j^{\hat{a}_2}}{|\hat{a}_2|}, \ \forall j \in \{1, 2, \cdots, |\mathcal{S}|\}$$

Then the comparison of pattern similarity scores for these two tools is only determined by the preliminary answer $\hat{a}$, pattern set size $|\mathcal{S}|$ and λ, with no sensitivity to the length of tool responses.

A.3 Mock Pattern

When dealing with a large tool library, iterating through all tools for true responses is inefficient and some tools may not have textual responses to encode. Conversely, through the utilization of pattern scores, we can set certain tools to generate mock responses with corresponding mock patterns during the tool grounding process, eliminating the requirement for actual execution, thereby reducing the GEAR's time complexity and generalizing it to various types of tools. In the experiment section §5, we test the efficiency and generalizability of mock patterns for tool grounding by adding `Sleep` and `Movement Controller` to the tool library.

B Implementation Details

B.1 Hyperparameters

To avoid bias toward to pattern similarity score, we use add-one smoothing and set $\lambda = 1$. Additionally, based on our experiment, we observed that the mean of pattern similarity score is consistently three times greater than the mean of the semantic score. In order to achieve a proper balance between these two scores, we set $\gamma = 0.75$ throughout the entire experiment.

B.2 Patterns

For 4 tools experiments, we use the following four patterns: $\mathcal{S}$ = {English token pattern: `e`, non-ASCII token pattern: `f`, number pattern: `n`, symbol pattern: `s`}. Because we believe these four basic patterns could cover a lot of language tools. Based on their frequency in the real corpus, we set their prior probabilities as follows: $\mathcal{P}$ = {`e`: 0.78, `f`: 0.18, `n`: 0.05, `s`: 0.02}.

For generalization experiments where the tool library size varies between 4 to 10, we consistently use the following prior pattern distribution: $\mathcal{P}$ = {`e`: 0.75, `f`: 0.15, `n`: 0.02, `s`: 0.02, Sleep Pattern: 0.02, Move pattern: 0.02, Time pattern: 0.02}.

B.3 Models

- `GPT-J` is from `https://huggingface.co/EleutherAI/gpt-j-6b`
- `GPT-Neo` is from `https://huggingface.co/EleutherAI/gpt-neo-1.3B`
- `MiniLM` is from `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`
- `MPNet` is from `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`

B.4 Tasks and Datasets

We use the following 12 datasets from 6 downstream tasks in our main experiment, plus 2 extra datasets (SQuAD (Rajpurkar et al., 2016) and Trex (Elsahar et al., 2018)) in Table 3. To keep the evaluation costs manageable, we use 1K instances per dataset.

- **Arithmetic (Arithm)**: We evaluate on ASDiv (Miao et al., 2020), GSM8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021) datastes. Given the arithmetic nature of these datasets, we expect successful grounding in `Calculator` tool should improve their performance.
- **Machine Translation (MT)**: We use IWSLT-2017 (Cettolo et al., 2017) dataset to evaluate the utility of successful grounding to the `MT` tool. The input data consists of an English prompt and a non-English context in Simplified Chinese, Arabic, German, French, Japanese, or Korean. We utilize diverse English prompts for English translation requests (e.g., "How do you say ... in English", "Speak ... to English", etc.). We sample 1K instances for each source language.
- **Open-domain QA (ODQA)**: We experiment with NQ-Open (Lee et al., 2019), WebQS (Berant et al., 2013), and TriviaQA (Joshi et al., 2017), since open-domain questions require external knowledge, successful grounding of these tasks to `Wiki` tool improve their performance.
- **Commensense QA (CSQA)**: To investigate the benefit of utilizing the `QA` tool, we evaluate all baselines on CommonsenseQA (Talmor et al., 2019), COPA (Roemmele et al., 2011), and SocialIQA (Sap et al., 2019). Those datasets require the model to perform commonsense reasoning for a given context and select the answer from a variety of choices.
- **Multilingual QA (MLQA)**: MLQA (Lewis et al., 2020) is a hard multilingual question-answering benchmark, expecting `Multilingual QA` to tackle such problem. Each instance includes an English context and a query presented in Arabic, German, Spanish, Hindi, Vietnamese, or Chinese. We randomly sample 1K instances for each language.

- **Timezone Conversion**: we create this dataset programmatically by iterating over all combinations of time zones, randomly-generated numbers which are verbalized into the natural language via real querying scenarios. Specifically, we set 5 querying templates and 3 time formats, combining them with randomly selected timezones to construct the dataset. Here are two examples:

> *My friend is in Cordoba, and I am in Madeira. If it is 2016-07-14 08:24:07 here, what time is it there?*
>
> *I want to make a call to someone. He is in Johannesburg, and I am in Pitcairn. If it is May 16 2023 10:31:14AM here, what time is it there?*

  Successfully grounding to the `Timezone Converter` should improve the performance of this task.

Evaluation Metrics. For the Arithmetic task, we convert all English numerals to their numerical equivalents and then pick the last number as the answer.[7] These are not needed when using `Calculator` tool, as it always outputs a single number. Ultimately, we compute an exact match accuracy between the resulting numbers and gold answers. For ODQA and MLQA tasks, following (Schick et al., 2023), we verify if the generated output contains the gold answer. For the CSQA task, we compute the accuracy as the ratio of accurately selected outputs. For the MT task, the translation quality is evaluated using a BLEU (as percentage).

[7] For zero-shot or few-shot baseline the overall answer typically appears after the rationales.

C Tools

We prioritize two factors for choosing tools: 1) whether they could compete with others 2) whether their function is naturally beyond the capability of any LLM.

C.1 Basic Tools

Description and usage prompts for each basic tool are provided in Table 15

- `QA`: Our question-answering system is based on an external language model specially trained for answering questions. We utilize ChatGPT in our experiment, renowned for its performance in comprehending and reasoning with human language.
- `Calculator`: The `Calculator` is built from the Python built-in function *eval*, which supports four fundamental arithmetic operations with priorities that can be specified using brackets. The output is rounded to three decimal places.
- `MT`: The core of our machine translation tool is the Google Translate API. It accepts two input arguments: the text to be translated and the target language.
- `Wiki`: The last basic tool employed in our experiment is the Wikipedia Search (Wikisearch) engine. It returns wiki paragraphs in response to queries. This tool advances models by supplying external factual knowledge and its returned output is more formal and informative than that of `QA`. In our experiment, we use ColBERTv2 (Santhanam et al., 2022) as the search retriever to index relevant information.

C.2 Novel Tools

For the selection of novel tools, we follow these two factors: whether they could compete with existing tools or whether their function is naturally beyond the capability of any LLM. Consequently, we add the following six tools:

- `Logarithmic Calculator` and `Exponential Calculator`: These two tools aim to solve logarithm and exponential problems and serve as competitors to the `Calculator` tool.
- `Multilingual QA`: We compose `MT` and `QA` tools to form the `Multilingual QA` pipeline. It involves two steps: translating the query to the target language using `MT`, and passing the context and translated query to the `QA` to find the final answer.
- `Timezone Converter`: This tool is implemented by the Python *pytz* library. It converts a time from one time zone to another. Such a task is also solvable by the QA tool but not accurately. Therefore, we want to assess the success rate of grounding the most appropriate tools for such endeavors.
- `Sleep`: This tool suspends the entire program for a specified duration. This tool is intended to test the mock response functionality for our system. We do not expect the program to sleep during the tool grounding procedure; a mocked response is sufficient. However, once selected, this tool should perform its intended function.
- `Movement Controller`: This tool instructs a robot to move a specified distance in a chosen direction. Similarly to `Sleep`, this tool is used for testing the mock response for grounding tools with non-textual outputs. During the grounding process, its returned response is a mock text: "`Robot is moving forward for {} meters`".

Models	ASDiv	GSM8K	SVAMP	NQ-Open	WebQA	TriviaQA
ART$_{cs}$ (MiniLM/GPT3$_{davinci-003}$)	**86.7**	69.7	77.3	56.7	17.7	61.0
GEAR (GPT-Neo/GPT3$_{davinci-003}$)	74.9 (-11.8)	**71.1 (+1.4)**	**79.9 (+2.6)**	53.8 (-2.9)	**23.6 (+5.9)**	**62.5 (+1.5)**

Table 8: Comparing GEAR with ART (Paranjape et al., 2023) on Arithmetic and Open-domain QA tasks

Algorithm →	Zero-shot	Few-shot	ART$^{*}_{llm}$	GEAR
Grounding Model →	–	–	GPT-Neo	GPT-Neo
Execution Model →	GPT-3	GPT-3	GPT-3	GPT-3
Datasets ↓				
ASDiv	78.7	75.3	37.0	74.9
GSM8K	62.4	69.9	14.7	71.1
SVAMP	75.4	73.7	21.3	79.9
↳ **Average (Arithm)**	72.2	73.0	24.3	**75.3**
IWSLT(cn)	43.1	30.1	19.2	39.2
IWSLT(ar)	47.2	41.1	16.1	41.8
IWSLT(de)	51.6	40.8	25.0	51.0
IWSLT(fr)	55.8	42.7	25.9	55.0
IWSLT(ja)	31.4	28.6	13.2	28.8
IWSLT(ko)	37.9	31.3	16.5	36.5
↳ **Average (MT)**	**44.5**	35.8	19.3	42.0
NQ-Open	58.0	66.1	24.0	53.8
WebQS	24.9	28.1	11.2	23.6
TriviaQA	54.9	70.4	24.3	62.5
↳ **Average (ODQA)**	45.9	**54.9**	19.8	46.6
CSQA	74.7	75.6	5.0	70.1
COPA	45.5	33.7	0.3	36.7
SocialIQA	56.8	64.8	1.2	59.5
↳ **Average (CSQA)**	**59.0**	58.0	2.2	55.4

Table 9: Downstream task performance result. Evidently, GEAR**-augmented** GPT-3 **achieves competitive results with** GPT-3 **few-shot and ART**, both of which provided with task-specific demonstrations for solutions.

D Downstream Performance

Results for GPT-3 baselines can be seen in Table 9. For MT and Commensense QA tasks, even the few-shot performance is lower than zero-shot, we hypothesize that this is because the GPT-3 model has seen those datasets during the pretraining and memorized them.

A comparison of GEAR with ART (Paranjape et al., 2023) on Arithmetic and Open-domain QA tasks is provided in Table 8. The downstream accuracy of GEAR-augmented GPT-3 is only slightly higher than those of ART-augmented GPT-3, because according to Table 4, ART achieves at least 90% grounding accuracy on most Arithmetic and Open-domain QA datasets. However, it is worth noting that ART requires in-domain demonstration for each task/dataset while GEAR does not.

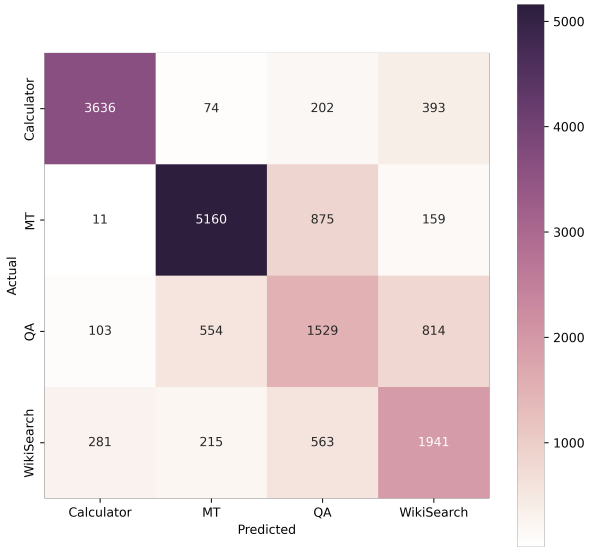


Figure 6: Confusion matrix of grounding results of four basic tools. Each number represents the number of examples being grounded to the tool.

Dataset (w/10 Tools)	**Target Tool**	GEAR	ART^*_{cs}
Average (Arithm)	`Cal`	74.0	97.2
Average (MT)	`MT`	**80.5**	78.5
Average (ODQA)	`Wiki`	**40.7**	21.1
Average (CSQA)	`QA`	**33.4**	22.8
Average (MLQA)	`MLQA`	**54.4**	17.6
Timezone Conversion	`TZ Conveter`	**96.4**	95.0

Table 10: Tool grounding accuracy for 6 downstream tasks with a 10-tools library (§5.2). GEAR **using** `GPT-Neo` **outperforms** ART^*_{cs} **using** `MPNet` **with the cosine similarity strategy on 5 out of 6 tasks.**

E Grounding Performance

According to Figure 6, it is clear that `Calculator` and `MT` tools have no strong competitors on Arithmetic and MT tasks, while `QA` and `Wiki` tools are more likely to compete with each other on CommonsenseQA and Open-domain QA tasks. This is due to the functional overlap of these two tools on open-ended NLP tasks.

GEAR **is more generalizable than retrieval-based baselines** We compare GEAR with two retrieval-based baselines: Okapi BM25 (Robertson et al., 1995) and KNN (Fix and Hodges, 1989) with 50 training examples for each tool under the 4-tools library. Like GEAR, BM25 is a general-purpose approach that does not need supervision. However, from Table 11, the grounding accuracy of BM25 is smaller than GEAR's (`GPT-Neo` version) on 13/15 datasets. All MT tasks get a 0% accuracy from BM25 since their inputs contain non-ASCII tokens, which are not accounted for in the description of the MT tool. Although the performance of KNN is generally higher than GEAR on MT and Open-domain tasks, it requires training and is easily overfitting, which hinders its generalizability to low-resource tasks that utilize novel tools without sufficient labeled data.

GEAR **is generalizable to novel tasks** We further evaluate GEAR's generalizability to novel tasks using MLQA (Lewis et al., 2020) and Timezone Conversion datasets. From Table 10, GEAR achieves 54.4% and 96.4% grounding accuracy on these two novel tasks with a 10-tools library. It outperforms ART^*_{cs} on 5 out of 6 tasks, revealing its strong generalizability to both large tool libraries and novel tasks.

Algorithm → Datasets ↓	BM25	KNN	GEAR(GPT-Neo)
ASDiv	0.5	66.5	83.1
GSM8K	0.2	58.7	83.0
SVAMP	1.2	75.1	89.0
↳**Average (Arithm)**	0.6	66.8	**85.0**
IWSLT(cn)	–	100	84.1
IWSLT(ar)	–	99.7	66.6
IWSLT(de)	–	80.0	96.9
IWSLT(fr)	–	84.6	96.6
IWSLT(ja)	–	100	72.4
IWSLT(ko)	–	100	82.2
↳**Average (MT)**	–	**94.1**	83.1
NQ-Open	76.2	73.4	63.0
WebQS	45.4	55.9	65.6
TriviaQA	62.5	83.3	54.3
↳**Average (ODQA)**	61.4	**70.9**	61.0
CSQA	24.6	75.8	77.1
COPA	0.6	32.9	41.3
SocialIQA	14.5	57.2	75.7
↳**Average (CSQA)**	13.2	55.3	**64.7**

Table 11: Tool grounding accuracy for 4 downstream tasks with a 4-tools library ("–" denotes 0). GEAR **with** GPT-Neo **consistently achieves high grounding performance compared to BM25 and KNN.**

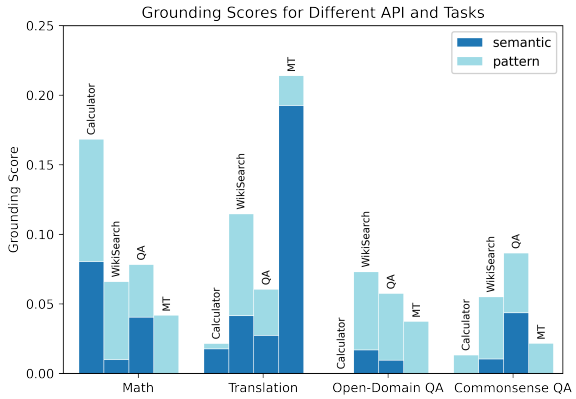


Figure 7: The average similarity scores for different tasks and tools. Clearly, **the semantic and pattern scores (already weighted by γ) collaboratively and accurately identify tools for the four basic tasks.**

F Ablation Study

For the 4-tools library, we plot the average final grounding score for each task and tool in Figure 7. Notably, neither the semantic nor the pattern similarity score dominates the query-tool grounding on most tasks, but they collaborate with each other to correctly identify the tools.

G GEAR Augmented Chatbot

Because GEAR does not require extra task-specific demonstrations, one of its practical applications is that it can be integrated into any chatbot. To validate it, we create a GEAR augmented chatbot using ChatGPT as the execution LLM and conduct a survey experiment.

Figure 8 illustrates the differences between GEAR-augmented chatbot and a normal chatbot and how GEAR interacts with a LM in a dialogue setting. For each user query, we first prompt the LM to determine if a tool usage is necessary. If true, the original query will be sent directly to GEAR, and GEAR will return the response from the selected tool as well as the tool name and confidence score for selecting that tool. This information is then processed by the LM to generate a more natural, user-friendly response. Figure 9 provides examples of how our GEAR augmented chatbot works. We equip it with the following six tools: Weather Search, Location Search, Image Generation, Current Time-Timezone Converter Pipeline, Wikipedia Search and Machine Translation.

We surveyed 50 individuals about the use of our GEAR-augmented chatbot. The evaluators first use ChatGPT-based chatbot for two weeks, then switch to a GEAR-augmented chatbot for the next two weeks. After fully experiencing these two chatbots, they are asked to complete the survey (Table 12) which contains four types of questions regarding tool grounding performance and final answer quality. Participants are unpaid and their feedback is unmodified.

The survey reveals that 76% of users agree that integrating tool usages makes the chatbot more useful and fascinating, and more than 90% of queries grounded correct tools. Image generation and weather search tools are the most popular tools among the 6 tools, with more than 50% of users employing them to solve problems. Regarding response quality, our survey indicates that an average of 78.4% of questions are answered to the user's satisfaction, a 16.9% increase in satisfaction compared to the previous chatbot that lacks the tool utilization functionality. The Current Time-Time Zone Convertor Pipeline has the highest accuracy, at 100%, while the Machine Translation tool has the lowest quality, with a satisfaction ratio of only 50.5%. We infer that the performance of the Google Translate API may not be adequate to satisfy the needs of our evaluators, given that most of them are translating extremely complex sentences between English, Japanese, and Chinese.

In summary, GEAR substantially improves users' experience on ChatGPT, and it also has excellent generalization capabilities to novel tools. Note that these novel tools lack training data, but with GEAR and just a few words of tool description and usage examples, they can be easily integrated into a chatbot to provide precise and reliable answers to users.

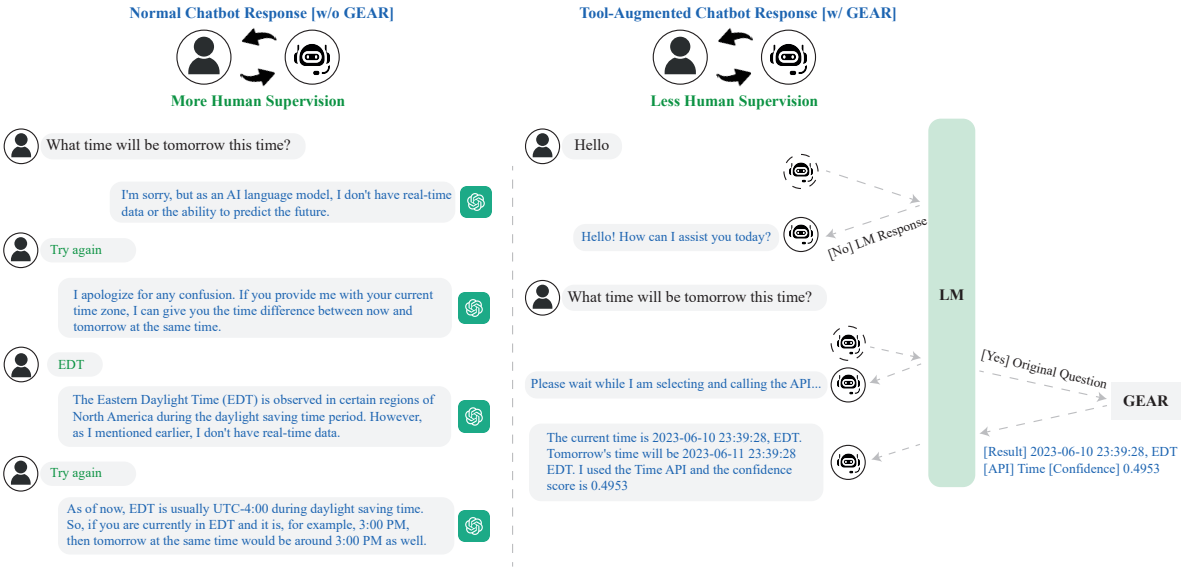
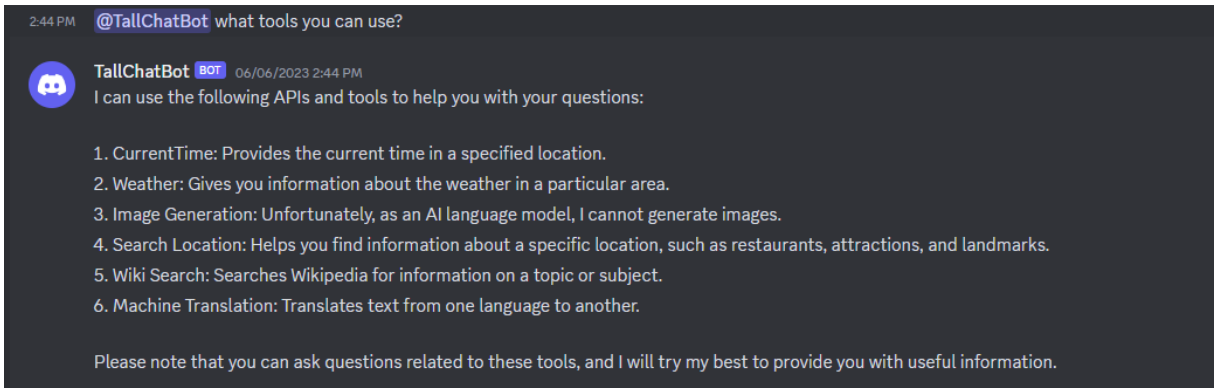
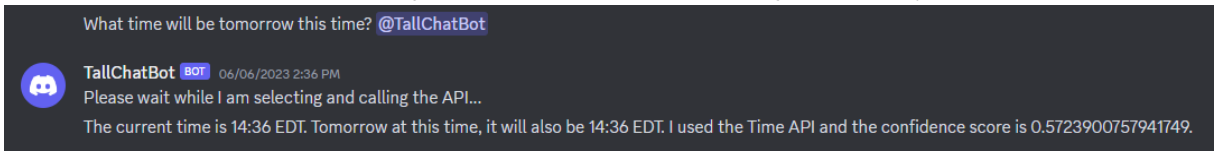


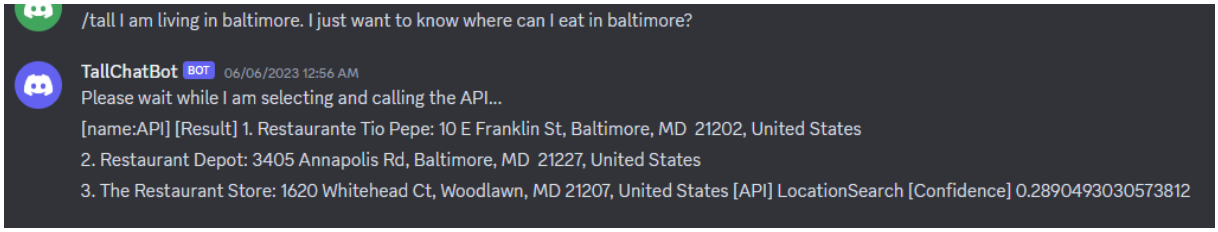
Figure 8: A comparison between the performance of ChatGPT and GEAR augmented chatbot. GEAR requires minimal human supervision, excels in numerous tool-solvable tasks, and offers interpretable confidence scores for users.



(a) GEAR augmented chatbot screenshot illustrating its tool library.



(b) GEAR augmented chatbot screenshot of using the Time tool.



(c) GEAR augmented chatbot screenshot of using the Location Search tool.

Figure 9: Screenshots of GEAR augmented chatbot using various tools. Using the command `/GEAR` to ask GEAR chatbot to output tool response directly without going through the ChatGPT. While the command `@TallChatBot` enables a normal conversation where GEAR interacts with ChatGPT to provide more human-readable answers.

Survery Question	Question Type	Answer
How would you rate your overall experience with GEAR-augmented chatbot?	rating scale	0-10
Do you think GEAR-augmented chatbot has become smarter compared to the previous version?	rating scale	0-10
Do you think GEAR-augmented chatbot has become more helpful than the previous one?	rating scale	0-10
How accurate do you think the answers of the older bot are?	rating scale	0-10
How accurate do you think the answers of the new version bot are?	rating scale	0-10
Have you noticed that the chatbot is using external tools to help you?	Likert scales	yes or no
How would you rate the chatbot's accuracy in choosing the right tool to answer your query?	rating scale	0-10
Can you recall a situation where the chatbot chose the wrong tool for your query? If so, please describe it briefly.	open-ended	open-ended
Have you ever instructed the chatbot to use a different tool for your query, or did the chatbot automatically choose a different tool because you weren't satisfied with the results?	Likert scales	yes or no
Will the chatbot be able to switch to the right tool based on your instructions?	Likert scales	yes or no
When a chatbot uses an external tool, how would you rate its response accuracy?	rating scale	0-10
Can you recall any instances where the chatbot used external tools to produce output errors or didn't meet your expectations? If so, please describe it briefly	open-ended	open-ended
What tools of chatbots have you used?	multiple-choice	multiple-choice
How would you rate the accuracy of the output generated by the chatbot using the Time tool?	rating scale	0-10
How would you rate the accuracy of the output generated by the chatbot using the Wikisearch tool?	rating scale	0-10
How would you rate the accuracy of the output generated by the chatbot using the Weather Lookup tool?	rating scale	0-10
How would you rate the accuracy of the output generated by the chatbot using the Location Search tool?	rating scale	0-10
How would you rate the accuracy of the output generated by the chatbot using the Image Generation tool?	rating scale	0-10
How would you rate the accuracy of the output generated by the chatbot using the Machine Translation tool?	rating scale	0-10
Please provide any additional feedback or suggestions you have for improving GEAR-augmented chatbot performance.	open-ended	open-ended
Overall Score you want give to the GEAR-augmented chatbot	rating scale	0-100

Table 12: Survey Questions

H Prompts

Table 13 provides examples of API calls and outputs for each tool
Table 14 shows task-specific demonstrations used for the few-shot baseline in the experiment
Table 15 presents the description and usage example of each basic tool.

Tool	Example API Call	Example Output
Question Answering	QA("What century did the Normans first gain their separate identity?")	The Normans first gained their separate identity in the 11th century.
Calculator	Calculator(2 + 4)	6
Machine Translation	MT("太多东西要在这18分钟内讲述了。", "en")	There are too many things to be described in this 18 minutes.
Wikipedia Search	WikiSearch("Lord Of The Flies")	Lord of the Flies (song) "Lord of the Flies" is an Iron Maiden single and second track on their 1995 album "The X Factor".
Multilingual QA	MultilingualQA("question: 《街机游戏街头霸王II》的游戏机上有多少用于控制角色的圆形物体? context: For example, the six button layout of the arcade games Street Fighter II or Mortal Kombat cannot be comfortably emulated on a console joypad, so licensed home arcade sticks for these games have been manufactured for home consoles and PCs.")	Six
Exponential	Pow(2, 3)	8
Logarithm	Log(2, 8)	3
Timezone Converter	TimezoneConverter("2022-01-02 22:00:00", "Asia/Shanghai", "America/New_York")	2022-01-02 09:00:00
Sleep	Sleep(20)	Sleep for 20 seconds (*Mock Response*)
Movement Controller	RobotMove(0.3)	Robot is moving forward for 0.3 meters (*Mock Response*)

Table 13: Examples of API Calls and Outputs for Each Tool

Task	Demonstration
Math	You are the Calculator tool. Your task is to answer the questions that contain numbers and require arithmetic operations, including addition, subtraction, multiplication, division. Here are some examples: Input: There were 86 pineapples in a store. The owner sold 48 pineapples. 9 of the remaining pineapples were rotten and thrown away. How many fresh pineapples are left? Output: There are total 86 pineapples. 48 pineapples are sold out, so there are 86 - 48 pineapples now. 9 of the remaining are thrown away, so there are 86 - 48 - 9 pineapples. That is 29 pineapples.
Commonsense Reasoning	You are the Question Answering tool that answers questions by reasoning and commonsense knowledge. Here are some examples: Input: The women met for coffee. What was the cause of this? A: The cafe reopened in a new location. B: They wanted to catch up with each other. Output: Considering the options, the more likely cause for the women meeting for coffee would be B: They wanted to catch up with each other. Meeting for coffee is often chosen as a way to have a relaxed and informal conversation, providing an opportunity for friends or acquaintances to reconnect and share updates about their lives.
Open-domain Question Answering	You are the Wikipedia Search tool that is to look up information from Wikipedia that is necessary to answer the question. Here are some examples: Input: The colors on the flag of Ghana have the following meanings: green for forests, and gold for mineral wealth. What is the meaning of red? Output: The color Red commemorates those who died or worked for the country's independence.
Machine Translation	You are the Machine Translation tool that is used for translating text from one language to another. Here are some examples: Input: How do I ask Japanese students if they had their dinner yet? Output: ごをもう食べましたか。

Table 14: Example of Various Task Demonstrations for Few-Shot Baselines

Tool	Description	Few-Shot Prompt
Calculator	Calculator API is used for answering questions that contain numbers and require arithmetic operations, including addition, subtraction, multiplication, division.	Calculator API is used for solving questions that require arithmetic operations, including addition, subtraction, multiplication, division. You task is to rephrase the question prepended by the special token <Q>and generate Calculator API call prepended by <API>for solving that question. You can call the API by writing "[Calculator(formula)]" where "formula" is the arithmetical formula you want to solve. Here are some examples of Calculator API calls: Input: There were 86 pineapples in a store. The owner sold 48 pineapples. 9 of the remaining pineapples were rotten and thrown away. How many fresh pineapples are left? Output: <Q>There are total 86 pineapples. 48 pineapples are sold out, so there are 86 - 48 pineapples now. 9 of the remaining are thrown away, so there are 86 - 48 - 9 pineapples. <API>[Calculator(86 - 48 - 9)].
Question Answering	Question Answering API answers questions by reasoning and commonsense knowledge.	Question Answering API answers questions by reasoning and commonsense knowledge. You task is to rephrase the question prepended by the special token <Q>and generate QA API call prepended by <API>for solving that question. Here are some examples of API calls: You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of QA API calls: Input: What do people want to acquire from opening business? A: home B: wealth C: bankruptcy D: get rich Output: <Q>What do people want to acquire from opening business? A: home B: wealth C: bankruptcy D: get rich <API>[QA("What do people want to acquire from opening business? A: home B: wealth C: bankruptcy D: get rich")].
Wiki Search	Wikipedia Search API is to look up information from Wikipedia that is necessary to answer the question.	Wikipedia Search API is to look up information from Wikipedia that is necessary to answer the question. You task is to rephrase the question prepended by the special token <Q>and generate Wikipedia Search API call prepended by <API>for solving that question. You can do so by writing "[WikiSearch(term)]" where "term" is the search term you want to look up. Here are some examples of WikiSearch API calls: Input: The colors on the flag of Ghana have the following meanings: green for forests, and gold for mineral wealth. What is the meaning of red? Output: <Q>Ghana flag green means forests, Ghana flag gold means mineral wealth, what is the the meaning of Ghana flag red? <API>[WikiSearch("Ghana flag red meaning")].
Machine Translation	Machine Translation API is used for translating text from one language to another.	Machine Translation API is used for translating text from one language to another. You task is to rephrase the question prepended by the special token <Q>and generate MT API call prepended by <API>for solving that question. You can do so by writing "[MT(text, target_language)]" where "text" is the text to be translated and "target_language" is the language to translate to. Here are some examples of MT API calls: Input: How do I ask Japanese students if they had their dinner yet? Output: <Q>Translate "Did you have dinner yet" in Japanese <API>[MT("Did you have dinner yet?", "ja")].

Table 15: Descriptions and Usage Prompts of Four Basic Tools