Automated Creation of Reusable and Diverse Toolsets for Enhancing LLM Reasoning

Zhiyuan Ma[1], **Zhenya Huang**[1,2*], **Jiayu Liu**[1], **Minmao Wang**[3], **Hongke Zhao**[3], **Xin Li**[1,4]

[1]State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China
[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
[3]College of Management and Economics, Tianjin University
[4]iFLYTEK AI Research
{zhyma, jy251198}@mail.ustc.edu.cn, {wmmmm, hongke}@tju.edu.cn, {huangzhy, leexin}@ustc.edu.cn

Abstract

Augmenting large language models (LLMs) with tools significantly enhances their problem-solving potential across multifaceted tasks. However, current tools automatically created by LLMs often serve as a mere summary of specific problems or solutions, which face two main issues: 1) Low reusability: The tools are overly problem-specific and struggle to handle new problems. 2) Limited diversity: The toolsets are too narrow, limiting their application to address a broader range of different problems. In this paper, we propose the Knowledge-grounded Tool Creation with Evolution (KTCE) framework, which aims to craft reusable and comprehensive toolsets for LLMs in a two-stage process. In the first stage (Knowledge-based Tool Creation), we conceptualize tools as a form of executable domain knowledge and propose a *problem-knowledge-tool* paradigm. Specifically, we leverage LLMs to abstract "knowledge" from "problems" and create a three-layer knowledge tree of *topics*, *concepts*, and *key points*. This hierarchical structure serves as a foundation for inducing atomic "tools" from "knowledge", grounding them in fundamental concepts and enhancing their usability. In the second stage (Tool Evolutionary Search), we evolve the toolsets through several actions including tool selection, mutation, and crossover. This stage mimics the biological evolution process, aiding toolsets in discovering new tools or updating existing ones, thereby increasing the diversity of the toolset. Experiments on challenging mathematical/tabular/scientific reasoning tasks demonstrate that our approach achieves substantial accuracy improvements ranging from 6.23% to 18.49% on average. Moreover, in-depth analyses reveal the superior characteristics of our toolkit, including high reusability, high diversity, and high generalizability on cross-data/LLM performance with low complexity.

Code — https://github.com/zhymma/KTCE

1 Introduction

Equipping LLMs with tools enables them to accomplish tasks beyond their native capabilities, making it a promising approach for solving highly complex problems (Qin et al. 2024a; Qu et al. 2024; Gao et al. 2024; Zhao et al. 2024; Luo et al. 2024; Gong and Sun 2024). As a popular type of tool, Python functions have been widely investigated to solve computational problems. For example, using SymPy enables LLMs to perform symbolic computations (e.g., calculus) (Meurer et al. 2017), while using Pandas allows LLMs to perform table-related operations (e.g., data filtering and aggregation) (Lu et al. 2024). Building on these, researchers have explored automatically generating Python functions as tools (Wang et al. 2024b; Cai et al. 2024) and further enhancing LLMs' capabilities through calling them.

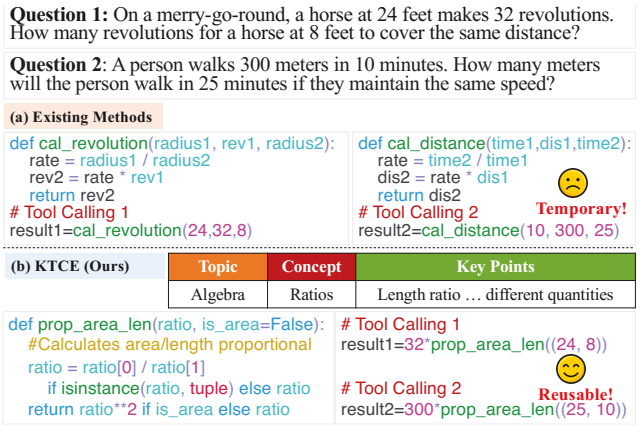

Figure 1: Comparison between (a) existing methods and (b) KTCE, illustrating that (a) existing methods require generating two temporary functions while (b) KTCE enables a reusable tool to address a class of problems.

However, as illustrated in Table 1, existing tools automatically created by LLMs still remain flawed in two key aspects. (Qian et al. 2023; Yuan et al. 2024; Wang, Neubig, and Fried 2024). Firstly, they are over problem-specific. For Question 1 (revolution proportional problem) in Figure 1(a), the generated `cal_revolution` tool can effectively solve Question 1 but fails to address the similar Question 2 (distance proportional problem), despite the fact that both of them require the same proportional knowledge to solve. Secondly, the toolsets often lack diversity, resulting in limited problem coverage. For example, tools created by TROVE (Wang, Neubig, and Fried 2024) can only be applied to less

*Corresponding author

Method	Reusability	Scale	Structure	Diversity	Optimization Capability
CREATOR	Low	-	-	-	-
CRAFT	Low	Large	Disorganized	Low	Low
TroVE	Low	Small	Disorganized	Limited	Low
KTCE (Ours)	High	Large	Organized	High	High

Table 1: Comparison of characteristics across different automatic tool creation methods.

than 15% of problems in the MATH dataset (Hendrycks et al. 2021), significantly restricting their applicability.

Behind these phenomena, we believe these issues of low tool reusability and limited toolset diversity stem from two critical oversights in current methods. Firstly, they neglect the importance of abstract knowledge in tool development. The essence of tools is a highly condensed knowledge. Therefore, the lack of explicit connection between tools and knowledge results in generated tools struggling to solve new problems that share the same knowledge points. Secondly, they completely rely on the problem itself to build static tools, neglecting the combination and expansion of these tools. Ultimately, these tools are limited in functionality and hard to be applied on different problems.

To address these challenges, in this paper, we propose a Knowledge-grounded Tool Creation with Evolution (KTCE) framework. It consists of two stages to automatically create reusable and diverse toolsets for enhancing LLMs' reasoning capabilities. In the first stage (**Knowledge-based Tool Creation**), we recognize that a tool is a form of executable, distilled abstract knowledge. While problems vary, the underlying knowledge is often interconnected (Liu et al. 2023b). Therefore, we establish a *problem-knowledge-tool* paradigm, which goes beyond existing methods by inducing tools based on the fundamental knowledge, rather than the problems themselves. Specifically, we automatically construct a hierarchical knowledge tree that contains three levels through knowledge extraction and clustering: *Topic* (e.g., "Algebra" in Figure 1(b)), *Concept* (e.g., "Ratios"), and *Key Points* (e.g., "Length ratio ..."). Based on this, we create atomic tools at *Key Points* level (e.g., `prop_area_len`) for each foundational *Topic-Concept* domain. In the second stage (**Tool Evolutionary Search**), we attempt to dynamically optimize the tools to further enhance the diversity of the toolset. We draw inspiration from Darwin's theory of evolution (Fisher 1999; Zhao et al. 2022) and Genetic Programming (GP) (Koza 1994) to implement tool evolution using LLMs by simulating biological genetics. Specifically, we design several actions including tool selection, mutation, and crossover to evolve the toolsets. To guide the evolutionary direction, we additionally design multi-perspective metrics and an optimization function to dynamically adapt KTCE's toolset. Finally, we built a KTCE-augmented Agent for practical reasoning, which leverages the generated toolset to solve problems.

We conduct experiments on 9 datasets across three challenging reasoning tasks: mathematics MATH (Hendrycks et al. 2021), table question-answering TabMWP (Lu et al. 2023), and science problems SCIBENCH (Wang et al. 2024a). We observe that KTCE consistently outperformed competitive baselines, improving reasoning accuracy by 6.23% to 18.49%. More importantly, our KTCE substantially improved the coverage to over 60% (compared to previous methods' 14.78%) and tripled tool usage frequency, indicating the creation of a highly reusable and widely diverse toolset. Further experiments validated that our toolset in KTCE achieved improvements in multiple aspects, including reduced solution complexity, and robust generalization across various LLM scales, architectures, and datasets.

2 Related Work

LLM Reasoning. Reasoning with LLM is a crucial benchmark for AI's problem-solving abilities and a key milestone towards AGI (Sun et al. 2024; Wu et al. 2024; Cheng et al. 2024; Zhang et al. 2024a). Recent LLM advancements have led to increased research in this area (Hong, Pang, and Zhang 2024; Xiao et al. 2024; Liu et al. 2024; Ni et al. 2024b,a; Zhang et al. 2024b; Wang et al. 2024c), with approaches falling into two categories. The first is language-based reasoning, which employs step-by-step intermediate steps and logical connections expressed in natural language (such as CoT (Wei et al. 2022), ToT (Yao et al. 2024) and GoT (Besta et al. 2024)) and DeAR (Xue et al. 2024), while the second is program-based reasoning, which harnesses the power of programming languages (Shi et al. 2024a), particularly Python, and code interpreters to perform precise computations and logical operations (such as PAL (Gao et al. 2023), PoT (Chen et al. 2023), ToRA (Gou et al. 2024), MAMMOTH (Yue et al. 2024)). Our research aims to enhance LLMs' reasoning by constructing tools and integrating them into code generation, enabling LLMs to tackle complex reasoning problems with executable domain knowledge.

Tool Utilization and Tool Creation. Tool utilization has gained significant attention for efficiently expanding LLM capabilities (Qin et al. 2024a; Shi et al. 2024b; Gou et al. 2024; Ma et al. 2024; Qin et al. 2024b; Schick et al. 2024). Tool-augmented LLMs significantly expand their capabilities by enabling information retrieval, complex computations, environmental interactions, efficiency improvements, and transparent reasoning processes. These models show promise in tackling complex problems similarly to human problem-solving strategies (Liu et al. 2023a). Given the scarcity and high costs of manual tools, researchers are focusing on how to automate the creation of high-quality tools to better assist LLMs in problem-solving. As shown in Table 1, current research can be categorized into two approaches: 1) Ad-hoc Tool Creation: CREATOR (Qian et al.

2023) generates temporary Python function tools for each problem. While this approach is straightforward, it shows low reusability as each tool is tightly coupled with a specific problem. 2) Toolset Formation: This approach includes VOYAGER (Wang et al. 2023), which develops Java tools through Minecraft interactions; CRAFT (Yuan et al. 2024), which abstracts Python tools from specific problem solutions, forming a large but disorganized toolset through validation and deduplication processes; and TROVE (Wang, Neubig, and Fried 2024), which iteratively samples and selects superior Python tools for a small toolbox with limited diversity. These methods typically focus on immediate problem-solving while neglecting the fundamental aspects of tool quality - reusability, scale, structure, diversity, and optimizability (as shown in Table 1). In contrast, our two-stage KTCE creates a reusable, widely applicable toolset that enhances LLM accuracy through tool invocation.

3 Knowledge-grounded Tool Creation with Evolution Framework

3.1 Preliminaries

Given a reasoning task with training dataset $D_{train} = \{(p_i, s_i)\}_{i=1}^{N}$, where p_i and s_i represent input problems (e.g., "What is b+c?") and corresponding solutions (e.g., "To convert this..."), KTCE aims to create a toolset $\mathcal{T}^*$ (we consider the Python functions as tools) based on D_{train} that can support LLMs' reasoning on testing set D_{test}. During this process, we have two primary objectives:

- **Tool Reusability**: Our tools can solve classes of problems rather than specific instances, which can be assessed by the frequency of calling each tool in T^* on D_{test}.
- **Toolset Diversity**: Our tools can cover a wide range of tasks, which can be reflected by the proportion of problems in D_{test} that can be solved by T^*.

To achieve these objectives, KTCE employs a two-stage framework as shown in Figure 2 and formally presented in Algorithm 1. Firstly, the **Knowledge-based Tool Creation** stage addresses reusability by generating initial toolset $\mathcal{T}$ from domain knowledge tree KT (Figure2(a)). Secondly, the **Tool Evolutionary Search** stage enhances diversity by iteratively expanding and optimizing $\mathcal{T}$ to produce the final toolset $\mathcal{T}^*$ (Figure 2(b)). These two stages directly correspond to the processes described in Sections 3.2 and 3.3, respectively. Finally, we design a KTCE-augmented Agent (Section 3.4) that enhances LLM's reasoning capabilities using $\mathcal{T}^*$. Detailed discussions of reusability and diversity metrics are provided in Section 4.1.

3.2 Stage 1: Knowledge-based Tool Creation

In this stage, we created the toolset T through three main steps: Knowledge Extraction, Knowledge Clustering, and Tool Creation, as shown in Figure 2(a). Different from previous methods where tool is used as a summary of the problem or solution (Qian et al. 2023; Yuan et al. 2024), we think that tools are abstracted domain knowledge that encapsulate problem-solving methods, transforming concepts into executable actions. Therefore, we establish a *problem-knowledge-tool* paradigm to transform domain knowledge into atomic tools, which are Python functions representing indivisible steps in the reasoning process.

Algorithm 1: The KTCE Framework

Input: Dataset D, toolset size k, max iter N
Output: Optimized toolset $\mathcal{T}^*$

Stage 1: Knowledge-based Tool Creation
$K \leftarrow \text{ExtractKnowledge}(D_{\text{train}})$
$T, C, KP \leftarrow \text{ClusterKnowledge}(K)$
$\mathcal{T} \leftarrow \{\}$
for $(t_i, c_j) \in T \times C$ **do**
  $F_{ij} \leftarrow \text{GenInitTools}(\text{GetKP}(t_i, c_j, T \times C \times KP))$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(t_i, c_j, F_{ij})\}$
end for
Stage 2: Evolutionary Search
for each (t_i, c_j) pair in $\mathcal{T}$ **do**
  Initialize $\mathcal{T}_{ij}, \mathcal{L}^{\text{best}}, \mathcal{T}_{ij}^{\text{best}}$ and metrics
  for $n = 1$ to N_{max} **do**
    $\mathcal{L}, \text{TES}, \text{TC}, \text{TA} \leftarrow \text{ToolEval}(\mathcal{T}_{ij}, D_{\text{train}}^{ij}, k)$
    if $\mathcal{L} < \mathcal{L}^{\text{best}}$ **then**
      $\mathcal{L}^{\text{best}}, \mathcal{T}_{ij}^{\text{best}} \leftarrow \mathcal{L}, \mathcal{T}_{ij}$
    end if
    $\mathcal{T}_{ij} \leftarrow \text{Evolution}(\mathcal{T}_{ij}, \text{TES}, \text{TC}, \text{TA})$
  end for
  Update $\mathcal{T}$ with $(t_i, c_j, \mathcal{T}_{ij}^{\text{best}})$
end for
return $\mathcal{T}^* \leftarrow \mathcal{T}$

Specifically, we first construct a domain knowledge tree KT through knowledge extraction. For each problem-solution pair (p_i, s_i) in D_{train}, we use LLMs to extract related knowledge in XML format, representing it as $K = \{(\textit{topic, concept, key points})\}$. For instance, a triplet might be ("Algebra", "Quadratic Equations", "Solving equations using ...") as shown in Figure 2(a).

These extracted knowledge elements are then consolidated into a three-level domain knowledge tree KT:

$$KT = T \cup C \cup KP, \tag{1}$$

where $T = \{t_1, t_2, ..., t_L\}$, $C = \{c_1, c_2, ..., c_N\}$, and $KP = \{kp_1, kp_2, ..., kp_M\}$ are sets of *topics, concepts, and key points* respectively. Using the BGE-M3 model (Chen et al. 2024) for semantic representations, we employ K-means clustering (Lloyd 1982) to group similar elements: Multiple concepts (e.g., "Quadratic Equations" and "Functions") are clustered into a topic (e.g., "Algebra"), and multiple key points are clustered into a concept. For each topic-concept pair, we cluster all associated key points to identify distinct subtasks, with the optimal number of clusters determined using the elbow method. Each key point cluster within a topic-concept pair represents a distinct *subtask*. For example, under the topic-concept pair "Algebra - Quadratic Equations", subtasks might include solving equations and finding the sum and difference of roots by Vieta's formula, as shown in Figure 2(a).

Following the knowledge tree construction, we then

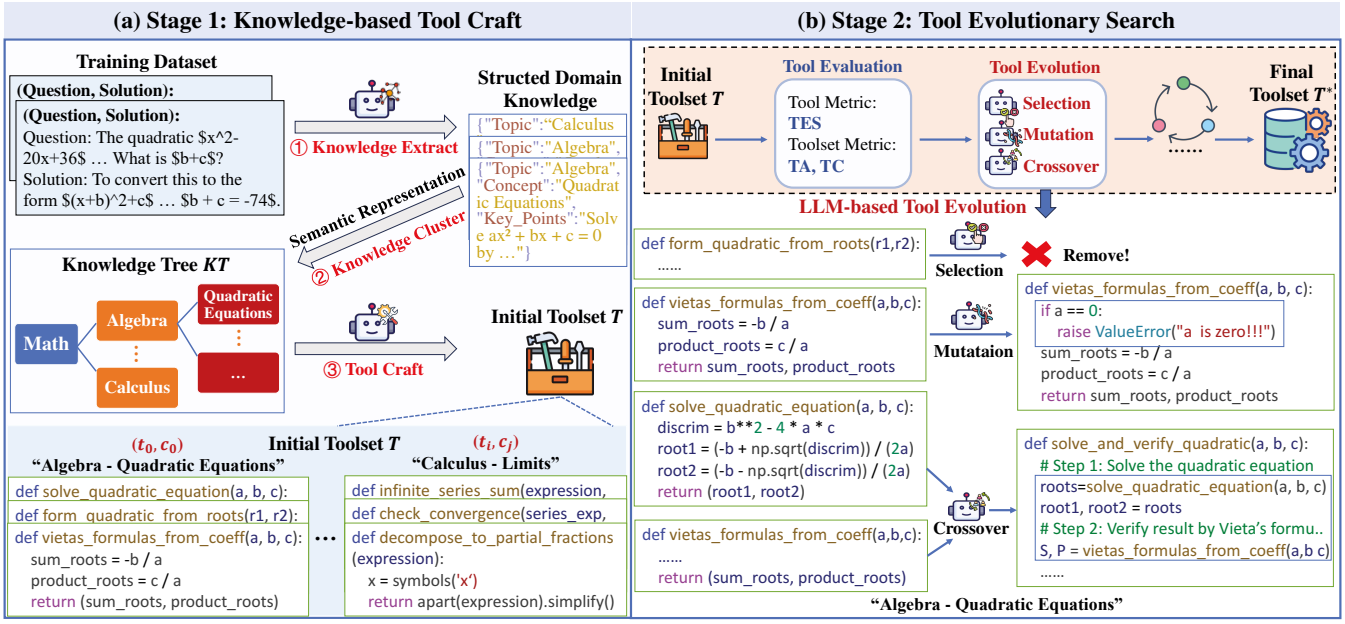

Figure 2: Our framework of KTCE, which consists of two main stages: (a) Stage 1 creates an initial toolset $\mathcal{T}$ based on a structured knowledge tree KT, and (b) Stage 2 optimizes the toolset into $\mathcal{T}^*$ through iterative tool evaluation and evolution.

prompt LLM to create atomic tools solving the subtasks in each topic-concept pair. As illustrated in Figure 2(a), this process generates functions such as `vietas_formulas_from_coeff` (for computing the sum and product of roots). The result is an initial toolset $\mathcal{T}$ corresponding to KT:

$$\mathcal{T} = \bigcup_{(t_i,c_j)\in T\times C} \{(t_i, c_j, F_{ij})\}, \tag{2}$$

where $F_{ij} = \{f_1, f_2, ..., f_{M_{ij}}\}$ is the set of funcs for topic t_i and concept c_j. Each function $f_k \in F_{ij}$ corresponds to a subtask solution based on kp_k in KP for the topic-concept pair (t_i, c_j). This structure ensures $\mathcal{T}$ directly reflects KP's organization, with each topic-concept pair having its own function set, and each function corresponding to a subtask.

This approach facilitates efficient management of large-scale tool collections while maintaining the inherent structure of domain knowledge. The structured toolset $\mathcal{T}$, with its hierarchical organization mirroring the knowledge tree KT, encompasses various knowledge points within the task domain. By breaking down complex reasoning tasks into atomic tools based on key points, our method enables flexible problem-solving reasoning.

3.3 Stage 2: Tool Evolutionary Search

In this stage, we completed the adaptive optimization of the initial toolset $\mathcal{T}$ through iterative Tool Evaluation and Tool Evolution, resulting in the final toolset $\mathcal{T}^*$, as shown in Figure 2(b). Although the initial toolset $\mathcal{T}$ from Stage 1 reflects the structure of domain knowledge, it may still have some limitations: 1) Lack of comprehensiveness: It may miss specialized tools for subtasks absent in the training set, limiting system's versatility. 2) Potential errors and inadequacies: Due to LLMs' inherent uncertainty, generated tools may have flaws. For instance, Figure 2(b) shows the `vietas_formulas_from_coeff` function missing the "$a = 0$" case. Such edge cases and boundary conditions can lead to incorrect reasoning results, resulting in unstable performance when LLMs handle complex reasoning tasks.

Given that these issues are inherent to static toolsets, dynamic optimization becomes crucial for improving tool effectiveness. However, a unique challenge arises as tools, unlike vectorized parameters in deep learning models, are Python functions that cannot compute gradients. Inspired by evolutionary biology and Genetic Programming (GP) (Koza 1994), we observe that mutation and crossover in biological evolution closely match our desired toolset updates.

Based on this insight, we propose an Evolutionary Search method to simulate this optimization process in a non-gradient manner, as shown in Figure 2(b). The main objective of this stage is to further enhance the diversity of the toolset. Specifically, we conceptualize tool f_k in $\mathcal{T}$ as individuals, toolset $\mathcal{T}$ as the initial population, and the dataset D_{train} as the environment. We leverage LLMs to simulate this evolutionary process to increase the diversity of the initial population and improve practical reasoning performance. We iteratively perform Tool Evaluation (simulate natural selection) and Tool Evolution (simulate selection, crossover, and mutation in evolution operations), with the former guiding the latter and controlling the iterative process. Finally, we can obtain the optimized final toolset $\mathcal{T}^*$.

Tool Evaluation. The Tool Evaluation phase assesses the performance of the population $\mathcal{T}$ on the environment D_{train}. This evaluation provides insights into the toolset's perfor-

mance, forming the foundation for the subsequent Tool Evolution phase. We use metrics and an optimization function that collectively reflect the adaptability of tools and toolset to the problem-solving environment, analogous to the fitness of individuals and populations in biological evolution.

To begin the evaluation process, for each F_{ij} with topic-concept pairs (t_i, c_j), we employ LLM to generate Python solutions by calling f_k in F_{ij} for sampled problems in D^{ij}_{train}. By executing these solutions, we obtain the accuracy of each program and the usage and accuracy of each tool.

We then quantify performance using several metrics. For individual tools, we measure: 1) Tool Invocation Frequency (TIF_{f_k}): represents the number of times f_k is called. 2) Tool Successful Invocation Frequency (TSIF_{f_k}): represents the number of times f_k is called and produces accurate results.

These metrics allow us to calculate a Tool Effectiveness Score ($\textbf{TES}_{f_k}$), defined as:

$$\text{TES}_{f_k} = 1 - \frac{\text{TSIF}_{f_k}}{\max(\text{TIF}_{f_k}, 1)}, \tag{3}$$

where a lower score indicates better performance of f_k.

For the toolset level, we evaluate: 1) Toolset Coverage (**TC**): represents the proportion of problems that using $\mathcal{T}$ to reason, i.e., the task coverage of $\mathcal{T}$ for D_{train}. 2) Task Accuracy (**TA**): measures the accuracy of reasoning using $\mathcal{T}$.

To guide the optimization process effectively, we introduce an optimization function $\mathcal{L}$ that integrates our three main objectives: tool reusability, toolset diversity, and ensuring reasoning accuracy. The function is defined as:

$$\begin{aligned}\mathcal{L} = \alpha \left(\sum_{f_k \in F_{ij}} \text{TES}_{f_k} + \max(0, k-n) \right) \\ + (\beta(1-\text{TC}) + \gamma(1-\text{TA})) \times |D^{ij}_{\text{train}}| \\ + \delta(n-k),\end{aligned} \tag{4}$$

where n is the current toolset size, k is the desired size, and α, β, γ, δ are weight coefficients. D^{ij}_{train} represents the subset of training dataset problems corresponding to the topic-concept pair (t_i, c_j).

By using these metrics and the comprehensive optimization function, we can effectively evaluate the performance of both individual tools and the entire toolset, providing crucial guidance for the subsequent Tool Evolution phase.

Tool Evolution. This phase leverages LLMs to implement three key mechanisms analogous to biological evolution (Fisher 1999; Koza 1994): selection, crossover, and mutation. These operations are applied to the toolset $\mathcal{T}$ based on the metrics and feedback obtained during the Tool Evaluation phase. Crucially, each operation is executed through carefully crafted prompts to the LLM, enabling it to generate updated Python functions in the toolset. The bottom part of Figure 2(b) demonstrates examples of the three operations.

Selection: Mimicking natural selection where only the fittest individuals reproduce, we prompt the LLM to retain effective tools and remove ineffective ones by analyzing the metrics and runtime information. For example, the tool `form_quadratic_from_roots` was rarely called during evaluation, resulting in a low TES. This indicates its presence is more noise than signal, leading to its removal.

Mutation: To ensure the reasoning accuracy of $\mathcal{T}$, we prompt the LLM to modify the function code by adjusting parameters, expanding functionality, and improving error handling to enhance adaptability. These targeted changes help explore the search space more thoroughly. For instance, in the tool `vietas_formulas_from_coeff`, error handling for edge cases was incorporated.

Crossover: To expand the diversity of $\mathcal{T}$, we generate new Python functions by combining features from two or more selected tools to create potentially more effective variants. We input the complete F_{ij} in $\mathcal{T}$, current TC scores, and uncovered subtasks, prompting the LLM to output new tools. For example, by combining two existing tools `solve_quadratic_equation` and `vietas_formulas_from_coeff`, a more powerful new tool `solve_and_verify_quaratic` can be created.

Throughout this evolutionary process, the optimization function L in Eq. (4) serves as a guiding metric. The goal is to minimize L by balancing tool usability, toolset coverage, and accuracy. If L does not decrease, a rollback mechanism is implemented to revert ineffective changes. Additionally, we employ early stopping when L shows no significant improvement over several iterations, ensuring computational efficiency. The process continues until either the stopping criterion is met or the maximum iterations are reached, resulting in the final optimized toolset T^*.

Our Evolutionary Search stage effectively addresses the initial toolset's limitations through LLM-driven iterative optimization. It systematically mirrors the biological evolution process to efficiently explore the solution space for enhancing both toolset diversity and reasoning accuracy. Moreover, this novel approach demonstrates the significant potential of combining LLM with evolutionary algorithms to create robust, adaptive, and self-improving agents.

3.4 KTCE-augmented Agent

After completing the two-stage process, we obtain the final optimized toolset $\mathcal{T}^*$. For practical reasoning, we develop a KTCE-augmented Agent (KA) to utilize tools from $\mathcal{T}^*$ for generating code solutions for input problems. The KA's process can be conceptualized as:

$$A = \mathcal{M}(p\ ;\ T^*, H) \tag{5}$$

where A is the final standardized answer, p is the input problem, T^* is the optimized toolset, H represents historical usage and error experiences, and $\mathcal{M}$ is the KA's problem-solving process.

Specifically, the KA operates through three key phases: Tool Retrieval, Solution Generation, and Result Formatting. In Tool Retrieval, it identifies the relevant (t_i, c_j) pair to access a targeted toolset F_{ij} from T^*, refined by historical data HH to ensure suitable tool selection. During Solution Generation, it leverages LLM to generate Python code that directly calls the retrieved tools, incorporating examples from HH through In-Context Learning (Dong et al. 2022). Finally, in Result Formatting, it executes the code to obtain intermediate results and transforms them into a standardized

answer format A for consistent evaluation. This structured approach enhances efficiency by directly accessing relevant tools without complex retrieval pipelines.

4 Experiments

4.1 Experimental Setup

Dataset and Evaluation We select three challenging reasoning tasks for evaluation.

- **Mathematical Reasoning**: We use the MATH dataset (Hendrycks et al. 2021) to test LLMs' text-based numerical reasoning. It has 7,500 training and 5,000 test problems across 7 categories of competition-level questions.
- **Tabular Reasoning**: The TabMWP dataset (Lu et al. 2023) is employed to assess LLMs' capability in processing structured tabular data and performing reasoning calculations. It includes 38,431 problems and 37,644 tables, with a test set of 1,000 problems.
- **Scientific Reasoning**: We utilize SCIBENCH dataset (Wang et al. 2024a) to examine numerical reasoning abilities in complex scientific contexts. This dataset contains 695 college-level problems from physics, chemistry, and mathematics. We randomly select 100 problems for testing and used the remainder for training.

Our evaluation incorporates a diverse range of answer types, including integers, multiple-choice questions, boolean values, mathematical expressions, and lists. This approach allows for a more comprehensive assessment of LLM reasoning capabilities across various problem formats.

We use answer accuracy (**Acc**) as the primary metric for evaluating model reasoning ability. Additionally, we employ task coverage (**Cov**), which describes the proportion of problems in the task that utilize tools. We also consider tool usage frequency (**# Freq**), which represents the average number of times each tool in the toolset is called. Finally, we measure toolset size (**# T-size**), which describes the number of functional tools in the toolset. These metrics collectively provide a comprehensive assessment of both the model's performance and the effectiveness of the toolset.

Baselines We compare KTCE with baseline methods:

- **Chain-of-Thought (CoT)** (Wei et al. 2022): Uses LLM to generate step-by-step thinking processes and logical relationships in natural language without tools.
- **Program-of-Thought (PoT)** (Chen et al. 2023): Utilize LLM to generate code to perform calculations and reasoning and run it through an external code interpreter.
- **Library** (Meurer et al. 2017): Augments PoT with external Python library functions (e.g., SymPy, Scipy).
- **Wolfram**: Enhances PoT by calling WolframAlpha API in Python for computational knowledge.
- **Creator** (Qian et al. 2023): Employs temporary tool creation for each problem, then generate solution code.
- **Creator (SR)** (Qian et al. 2023): Enhances Creator with self-refinement based on execution feedback.
- **CRAFT** (Yuan et al. 2024): Implements toolset formation by abstracting, validating, and deduplicating tools. Uses LLM for query generation and tool retrieval.
- **TROVE** (Wang, Neubig, and Fried 2024): Utilizes multiple sampling for tool creation, generating various tools and solutions and form a compact toolbox.

Implementation We implement KTCE, and all baselines using GPT-3.5-Turbo for its cost-effectiveness, speed, and code generation capabilities. In Section 3.2, we extract max 3 knowledge triplets K per problem. For Section 3.3, we sample up to 100 problems per topic-concept pair, with max 5 iterations. Early stopping is applied if L didn't decrease for 3 consecutive iterations. Desired toolset size k is 10. For KA (in Section 3.4) and all baselines, to better utilize the tools, we allow up to three sampling attempts until the solution code successfully compiles.

4.2 Main Results

Table 2 presents the reasoning accuracy across all datasets. Overall, KTCE generates the most accurate solutions for three tasks, demonstrating the effectiveness of KTCE-generated toolset and KA. For the seven types of MATH dataset, KTCE improves accuracy by 1.38% to 11.97% compared to TROVE, demonstrating how our knowledge-based tools can be effectively reused across different mathematical problem types. Additionally, it outperforms baseline using Python Library by up to a maximum of 10.2%. These results demonstrate KTCE's superior capability in creating high-quality mathematical tools, enabling LLMs to effectively tackle competition-level problems.

In tabular reasoning, KTCE is the only method to exceed 90% accuracy. For the more challenging scientific reasoning domain, it improves LLM performance by 6% compared to PoT where most other methods underperform PoT. These results demonstrate KTCE's unique advantage in leveraging domain knowledge - while tabular reasoning benefits from well-structured table operations, scientific reasoning showcases how KTCE effectively captures complex subject knowledge through our knowledge-grounded approach. This overcomes the limitation of previous methods in creating reusable tools across different tasks.

Building upon these observations, the experimental results in Table 3 further demonstrate KTCE's capability in creating reusable and diverse tools. The high coverage rates (64.50%, 82.90%, and 62.00% across three datasets) and increased tool usage frequency validate that our knowledge-based tool creation (Stage 1) successfully addresses the reusability limitation of previous methods by grounding tools in fundamental domain knowledge. Meanwhile, the larger and more diverse toolset compared to CRAFT confirms that our evolutionary search (Stage 2) effectively expands tool diversity, overcoming the narrow toolset problem in existing approaches.

In summary, KTCE addresses the core challenges through two-stage framework: knowledge-based creation enables tool reusability by grounding in domain knowledge, and evolutionary search ensures toolset diversity through systematic optimization. These advantages provide a solid foundation for enhancing LLMs with high-quality tools.

Type	Method	MATH							TabMWP	SCIBENCH
		Alg	Count	Geo	Int	Num	Pre.Alg	Pre.Cal		
Basic	CoT	49.12	29.75	22.34	14.62	33.33	53.85	16.85	73.50	27.00
	PoT	48.36	43.88	31.32	18.27	52.22	65.10	20.33	74.70	31.00
Tool-Aug.	Library	58.80	51.90	33.40	**29.90**	57.22	68.66	22.16	78.20	30.00
	Wolfram	55.27	37.76	28.60	20.49	34.81	61.77	26.92	68.00	26.00
Tool Creation	Creator	34.29	43.04	25.47	24.81	38.52	43.86	21.06	84.90	27.00
	Creator (SR)	50.38	48.73	28.18	28.90	48.70	62.34	19.78	87.80	32.00
	CRAFT	53.33	42.62	22.96	25.25	41.67	69.35	20.51	77.00	28.00
	TROVE	57.03	52.00	30.06	26.02	45.93	66.02	19.96	65.00	25.00
Ours	KTCE	**69.00***	**53.38***	**40.29***	**29.90***	**57.96***	**73.02***	**31.68***	**90.00***	**37.00***

Table 2: GPT-3.5-Turbo reasoning accuracy on three challenging reasoning datasets (%). **Bold** values indicate the highest scores, underlined values indicate the second highest. *indicates statistical significance ($p < 0.05$).

Method	Metric	MATH	TabMWP	SCIBENCH
CRAFT	Cov	19.10%	30.50%	6.00%
	# Freq	1.02	1.69	0.27
	# T-size	936	180	22
TROVE	Cov	14.78%	55.00%	20.00%
	# Freq	0.56	1.47	0.37
	# T-size	1347	399	65
KTCE	Cov	**64.50%**	**82.90%**	**62.00%**
	# Freq	**3.10**	**4.37**	**0.45**
	# T-size	1317	222	199

Table 3: Comparison of toolsets created by various methods.

Method	Acc	Cov	# Freq	# T-size
KTCE	**53.14%**	**64.50%**	**3.10**	1317
w/o Stage 1	50.16%	9.16%	5.99	78
w/o Stage 2	50.50%	57.92%	2.14	1612
w/o Sel	51.72%	65.10%	2.04	1858
w/o CO	50.62%	59.86%	2.85	1278
w/o Mut	51.56%	64.42%	2.65	1402

Table 4: Ablation study results on the MATH dataset.

4.3 Ablation Study

We conduct ablation experiments on the MATH dataset to evaluate each KTCE component. From Table 4, we observe that removing entire stages ("w/o Stage 1" and "w/o Stage 2") decreases accuracy. Eliminating Stage 1 (Knowledge-based Tool Creation) drastically reduces the size and diversity of toolsets (only 78 tools with 9.16% coverage), indicating that the structured domain knowledge in Stage 1 is crucial for creating a comprehensive toolset. Removing Stage 2 (Tool Evolutionary Search) leads to decreased coverage and tool usage frequency despite an enlarged toolset, demonstrating that evolutionary optimization is necessary to ensure toolset effectiveness.

We then ablate the three operations in Stage 2: selection ("w/o Sel"), crossover ("w/o CO"), and mutation ("w/o Mut"). Each operation proves essential for toolset quality:

Method \ Metric	CC↓	HV↓	HE↓
PoT	2.32	66.36	247.83
PoT with Library	1.83	64.85	190.82
CRAFT	1.50	78.61	231.93
TROVE	1.57	82.79	235.88
KTCE	**1.49**	**44.10**	**119.71**

Table 5: Complexity of Python solutions in the MATH dataset for different methods.

1) Removing selection leads to redundant, ineffective tools, shown by decreased accuracy despite more tools. 2) Without crossover, new tools can't be generated through combination, limiting toolset diversity. 3) Removing mutation prevents necessary tool updates, reducing adaptation capability. These results demonstrate how each component contributes to KTCE's success - Stage 1 ensures comprehensive tool creation through knowledge structuring, while Stage 2's operations maintain toolset quality and diversity.

4.4 KTCE Analysis

Code Complexity Analysis Tools crucially allow LLMs to avoid generating complete solution code from scratch. To evaluate KTCE's performance in further simplifying reasoning, we compare the complexity of solution codes generated by KTCE and baselines on the MATH dataset. We employ three established code evaluation metrics:

- Cyclomatic Complexity (CC): Measures linearly independent paths in the source code.
- Halstead Volume (HV): Measures program size based on operands, operators, and length.
- Halstead Effort (HE): Estimates cognitive load for designing, writing, and understanding the program (McCabe 1976; Halstead 1977).

As shown in Table 5, KTCE produces solutions with the lowest complexity across all metrics. The reduced Cyclomatic Complexity indicates simpler control flow structures, while lower Halstead Volume and Effort suggest

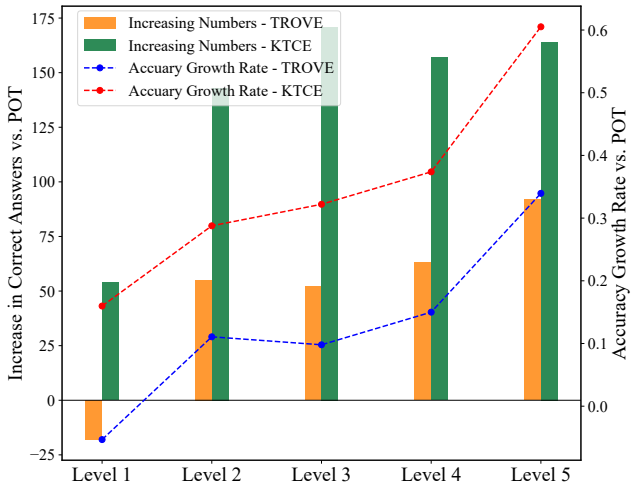

Figure 3: Comparison of Reasoning Accuracy of TROVE and KTCE w.r.t. Difficulty Level in the MATH dataset.

Model	Dataset	Method	Accuracy
GPT-3.5-Turbo	MATH → GSM8K	PoT	78.62%
		KTCE	**81.96%**
GPT-3.5-Turbo → GPT-4o-Mini	MATH	PoT	69.94%
		KTCE	**74.00%**
GPT-3.5-Turbo → DeepSeek-Coder	SCIBENCH	PoT	30.00%
		KTCE	**35.00%**

Table 6: Performance of KTCE toolset when generalized to different LLMs and datasets.

more concise and comprehensible solutions. These improvements stem from KTCE's ability to create reusable tools that encapsulate common problem-solving patterns, allowing LLMs to leverage accumulated knowledge rather than regenerating similar code repeatedly.

By building a comprehensive and reusable toolset through systematic knowledge extraction, KTCE effectively enables LLMs to focus on high-level reasoning rather than repetitive code generation. This knowledge-grounded approach significantly reduces solution complexity while substantially enhancing reasoning efficiency, as LLMs can concentrate on crucial problem-solving steps while reliably utilizing well-tested tools for implementation.

Reasoning Across Difficulty Levels To assess KTCE's robustness across difficulty levels, we analyze its performance on problems of varying complexity, as provided by the MATH dataset. Figure 3 shows the increase in correct reasoning instances (bars) and accuracy growth rate (line) relative to PoT. KTCE outperforms TROVE at all levels, with notable improvements on challenging problems. These results validate KTCE's effectiveness in enhancing reasoning capabilities across difficulty levels, particularly excelling at complex problems through its well-designed toolset.

To explore KTCE's generalizability, we apply the toolset created by GPT-3.5-Turbo across LLMs and datasets. From Table 6, we first observe that KTCE's knowledge-grounded tools, created from MATH dataset, effectively transfer to GSM8K with a 3.34% accuracy improvement (Cobbe et al. 2021). This cross-dataset success validates that our knowledge-based tool creation approach successfully captures fundamental mathematical concepts, enabling tools to generalize across different mathematical tasks. Secondly, to evaluate the impact of the constructed toolset on different models, we provide GPT-3.5-Turbo's tools to GPT-4o-Mini and DeepSeek-Coder on two datasets. When applied to GPT-4o-Mini on MATH, the accuracy improves by 4.06%, indicating potential across different model scales (OpenAI 2024). For DeepSeek-Coder on SCIBENCH, the accuracy rises by 5.00%. These findings demonstrate the cross-model robustness and generalizability of our approach.

These comprehensive results demonstrate KTCE's unique ability to create a reusable and diverse toolset that significantly enhances reasoning across diverse LLMs and domains without fine-tuning, effectively highlighting its versatility and wide applicability.

5 Conclusion and Future Work

In this paper, we have introduced Knowledge-grounded Tool Creation with Evolution (KTCE), a two-stage framework for creating reusable and diverse toolsets for LLMs. By combining Knowledge-based Tool Creation with Tool Evolutionary Search, KTCE substantially improved tool reusability and diversity, breaking through the limitations of previous methods and laying the foundation for large-scale LLM tool utilization. Experiments across challenging reasoning tasks demonstrated its effectiveness, outperforming baselines in accuracy and tool quality. The resulting toolset enhanced LLMs' problem-solving capabilities while reducing solution complexity. KTCE's approach was analogous to learning explicit knowledge from tasks, preserving model integrity while enabling easy extension to different LLMs. Moreover, the tool evolution process enabled the combination of different tools, discovering connections between various knowledge domains. In the future, we plan to explore more advanced tools such as multi-modal and embodied AI tools, and design better agent workflows for tool utilization. These advancements will contribute to KTCE's continued development in enhancing LLMs' capabilities across diverse tasks.

Acknowledgments

This research was partially supported by grants from the National Natural Science Foundation of China (No.62477044, 72471165, 62106246), and the Key Technologies R&D Program of Anhui Province (No.202423k09020039), and the Fundamental Research Funds for the Central Universities.

References

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. 2024. Graph of thoughts: Solving elaborate problems with large language

models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17682–17690.

Cai, T.; Wang, X.; Ma, T.; Chen, X.; and Zhou, D. 2024. Large Language Models as Tool Makers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; and Liu, Z. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Chen, W.; Ma, X.; Wang, X.; and Cohen, W. W. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.

Cheng, C.; Zhao, G.; Huang, Z.; Zhuang, Y.; Pan, Z.; Liu, Q.; Li, X.; and Chen, E. 2024. Towards Explainable Computerized Adaptive Testing with Large Language Model. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2655–2672.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Dong, Q.; Li, L.; Dai, D.; Zheng, C.; Wu, Z.; Chang, B.; Sun, X.; Xu, J.; and Sui, Z. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Fisher, R. A. 1999. *The genetical theory of natural selection: a complete variorum edition*. Oxford University Press.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, 10764–10799. PMLR.

Gao, S.; Shi, Z.; Zhu, M.; Fang, B.; Xin, X.; Ren, P.; Chen, Z.; Ma, J.; and Ren, Z. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 18030–18038.

Gong, Z.; and Sun, Y. 2024. Graph Reasoning Enhanced Language Models for Text-to-SQL. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2447–2451.

Gou, Z.; Shao, Z.; Gong, Y.; yelong shen; Yang, Y.; Huang, M.; Duan, N.; and Chen, W. 2024. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. In *The Twelfth International Conference on Learning Representations*.

Halstead, M. H. 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Hong, R.; Pang, X.; and Zhang, C. 2024. Advances in reasoning by prompting large language models: A survey. *Cybernetics and Intelligence*.

Koza, J. R. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4: 87–112.

Liu, J.; Huang, Z.; Ma, Z.; Liu, Q.; Chen, E.; Su, T.; and Liu, H. 2023a. Guiding Mathematical Reasoning via Mastering Commonsense Formula Knowledge. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1477–1488.

Liu, J.; Huang, Z.; Xiao, T.; Sha, J.; Wu, J.; Liu, Q.; Wang, S.; and Chen, E. 2024. SocraticLM: Exploring Socratic Personalized Teaching with Large Language Models. In *Advances in Neural Information Processing Systems*, volume 37.

Liu, J.; Huang, Z.; Zhai, C.; and Liu, Q. 2023b. Learning by applying: A general framework for mathematical reasoning via enhancing explicit knowledge learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 4497–4506.

Lloyd, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2): 129–137.

Lu, P.; Peng, B.; Cheng, H.; Galley, M.; Chang, K.-W.; Wu, Y. N.; Zhu, S.-C.; and Gao, J. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36.

Lu, P.; Qiu, L.; Chang, K.; Wu, Y. N.; Zhu, S.; Rajpurohit, T.; Clark, P.; and Kalyan, A. 2023. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*.

Luo, X.; Zhu, Q.; Zhang, Z.; Qin, L.; Zhang, X.; Yang, Q.; Xu, D.; and Che, W. 2024. Python is Not Always the Best Choice: Embracing Multilingual Program of Thoughts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 7185–7212.

Ma, Y.; Gou, Z.; Hao, J.; Xu, R.; Wang, S.; Pan, L.; Yang, Y.; Cao, Y.; and Sun, A. 2024. SciAgent: Tool-augmented Language Models for Scientific Reasoning. *arXiv preprint arXiv:2402.11451*.

McCabe, T. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4): 308–320.

Meurer, A.; Smith, C. P.; Paprocki, M.; Čertík, O.; Kirpichev, S. B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J. K.; Singh, S.; et al. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3: e103.

Ni, S.; Bi, K.; Guo, J.; and Cheng, X. 2024a. When Do LLMs Need Retrieval Augmentation? Mitigating LLMs' Overconfidence Helps Retrieval Augmentation. *arXiv preprint arXiv:2402.11457*.

Ni, S.; Bi, K.; Yu, L.; and Guo, J. 2024b. Are Large Language Models More Honest in Their Probabilistic or Verbalized Confidence? *arXiv preprint arXiv:2408.09773*.

OpenAI. 2024. GPT-4o-mini: Advancing Cost-Efficient Intelligence.

Qian, C.; Han, C.; Fung, Y.; Qin, Y.; Liu, Z.; and Ji, H. 2023. CREATOR: Tool Creation for Disentangling Abstract and Concrete Reasoning of Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 6922–6939.

Qin, Y.; Hu, S.; Lin, Y.; Chen, W.; Ding, N.; Cui, G.; et al. 2024a. Tool Learning with Foundation Models. arXiv:2304.08354.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; Zhao, S.; Hong, L.; Tian, R.; Xie, R.; Zhou, J.; Gerstein, M.; Li, D.; Liu, Z.; and Sun, M. 2024b. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; and Wen, J.-R. 2024. Tool Learning with Large Language Models: A Survey. *arXiv preprint arXiv:2405.17935*.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Shi, Z.; Gao, S.; Chen, X.; Feng, Y.; Yan, L.; Shi, H.; Yin, D.; Chen, Z.; Verberne, S.; and Ren, Z. 2024a. Chain of Tools: Large Language Model is an Automatic Multi-tool Learner. *arXiv preprint arXiv:2405.16533*.

Shi, Z.; Gao, S.; Chen, X.; Feng, Y.; Yan, L.; Shi, H.; Yin, D.; Ren, P.; Verberne, S.; and Ren, Z. 2024b. Learning to use tools via cooperative and interactive agents. *arXiv preprint arXiv:2403.03031*.

Sun, J.; Zheng, C.; Xie, E.; Liu, Z.; Chu, R.; Qiu, J.; Xu, J.; Ding, M.; Li, H.; Geng, M.; et al. 2024. A Survey of Reasoning with Foundation Models. arXiv:2312.11562.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*.

Wang, X.; Hu, Z.; Lu, P.; Zhu, Y.; Zhang, J.; Subramaniam, S.; Loomba, A. R.; Zhang, S.; Sun, Y.; and Wang, W. 2024a. SciBench: Evaluating College-Level Scientific Problem-Solving Abilities of Large Language Models. In *Forty-first International Conference on Machine Learning*.

Wang, Z.; Cheng, Z.; Zhu, H.; Fried, D.; and Neubig, G. 2024b. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

Wang, Z.; Neubig, G.; and Fried, D. 2024. TroVE: Inducing Verifiable and Efficient Toolboxes for Solving Programmatic Tasks. In *Forty-first International Conference on Machine Learning*.

Wang, Z.; Wang, P.; Liu, K.; Wang, P.; Fu, Y.; Lu, C.-T.; Aggarwal, C. C.; Pei, J.; and Zhou, Y. 2024c. A Comprehensive Survey on Data Augmentation. *arXiv preprint arXiv:2405.09591*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, L.; Li, Z.; Zhao, H.; Huang, Z.; Han, Y.; Jiang, J.; and Chen, E. 2024. Supporting Your Idea Reasonably: A Knowledge-Aware Topic Reasoning Strategy for Citation Recommendation. *IEEE Transactions on Knowledge and Data Engineering*.

Xiao, T.; Liu, J.; Huang, Z.; Wu, J.; Sha, J.; Wang, S.; and Chen, E. 2024. Learning to Solve Geometry Problems via Simulating Human Dual-Reasoning Process. In Larson, K., ed., *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 6559–6568. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Xue, S.; Huang, Z.; Liu, J.; Lin, X.; Ning, Y.; Jin, B.; Li, X.; and Liu, Q. 2024. Decompose, Analyze and Rethink: Solving Intricate Problems with Human-like Reasoning Cycle. In *Advances in Neural Information Processing Systems*, volume 37.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Yuan, L.; Chen, Y.; Wang, X.; Fung, Y.; Peng, H.; and Ji, H. 2024. CRAFT: Customizing LLMs by Creating and Retrieving from Specialized Toolsets. In *The Twelfth International Conference on Learning Representations*.

Yue, X.; Qu, X.; Zhang, G.; Fu, Y.; Huang, W.; Sun, H.; Su, Y.; and Chen, W. 2024. MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning. In *The Twelfth International Conference on Learning Representations*.

Zhang, J.; Wang, X.; Jin, Y.; Chen, C.; Zhang, X.; and Liu, K. 2024a. Prototypical Reward Network for Data-Efficient RLHF. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 13871–13884. Bangkok, Thailand: Association for Computational Linguistics.

Zhang, J.; Wang, X.; Ren, W.; Jiang, L.; Wang, D.; and Liu, K. 2024b. RATT: AThought Structure for Coherent and Correct LLMReasoning. *arXiv preprint arXiv:2406.02746*.

Zhao, H.; Wu, X.; Zhao, C.; Zhang, L.; Ma, H.; and Cheng, F. 2022. CoEA: A Cooperative–Competitive Evolutionary Algorithm for Bidirectional Recommendations. *IEEE Transactions on Evolutionary Computation*, 26(1): 28–42.

Zhao, Y.; Huang, Z.; Ma, Y.; Li, R.; Zhang, K.; Jiang, H.; Liu, Q.; Zhu, L.; and Su, Y. 2024. RePair: Automated Program Repair with Process-based Feedback. In *Findings of the Association for Computational Linguistics ACL 2024*, 16415–16429.