



AAAI-25 / IAAI-25 / EAAI-25
FEBRUARY 25 – MARCH 4, 2025 | PHILADELPHIA, USA



Automated Creation of Reusable and Diverse Toolsets for Enhancing LLM Reasoning

Zhiyuan Ma, Zhenya Huang, Jiayu Liu, Minmao Wang, Hongke Zhao, Xin Li

State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China

Institute of Artificial Intelligence, Hefei Comprehensive National Science Center

College of Management and Economics, Tianjin University



- **Introduction**

- KTCE Framework

- Experiments

- Conclusion



□ Tool Augmented Reasoning

- Tools (e.g., Python Library functions like Sympy) empower LLMs to tackle complex reasoning tasks by precise computation

□ Tool Creation

- Manually constructing tools requires substantial costs and the resources are limited in quantity.
- Many studies have automatically created tools in various fields (e.g., math) using Large Language Models (LLMs), and invoked them to enhance the reasoning ability of LLMs

Existing tool creation methods face limitations:

Low Reusability

- Tools are over problem-specific, which limits their ability to be reused for other tasks. (e.g., a "revolution ratio" tool fails on similar "distance ratio" tasks)

Limited Diversity

- Toolsets lack coverage for broader tasks. Tools created by TROVE can only be applied to less than 15% of problems in the MATH dataset

These limitations severely restrict the application of tool creation

Question 1: On a merry-go-round, a horse at 24 feet makes 32 revolutions. How many revolutions for a horse at 8 feet to cover the same distance?

Question 2: A person walks 300 meters in 10 minutes. How many meters will the person walk in 25 minutes if they maintain the same speed?

(a) Existing Methods

<pre>def cal_revolution(radius1, rev1, radius2): rate = radius1 / radius2 rev2 = rate * rev1 return rev2 # Tool Calling 1 result1=cal_revolution(24,32,8)</pre>	<pre>def cal_distance(time1,dis1,time2): rate = time2 / time1 dis2 = rate * dis1 return dis2 # Tool Calling 2 result2=cal_distance(10, 300, 25)</pre>
---	---



Temporary!

(b) KTCE (Ours)	Topic	Concept	Key Points
	Algebra	Ratios	Length ratio ... different quantities

<pre>def prop_area_len(ratio, is_area=False): #Calculates area/length proportional ratio = ratio[0] / ratio[1] if isinstance(ratio, tuple) else ratio return ratio**2 if is_area else ratio</pre>	<pre># Tool Calling 1 result1=32*prop_area_len((24, 8)) # Tool Calling 2 result2=300*prop_area_len((25, 10))</pre>
---	--



Reusable!

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.

□ These limitations stem from two oversights:

□ **Low Reusability**

- The neglect of the importance of abstract knowledge in tool development. Tools are highly condensed knowledge, and without an explicit connection between tools and knowledge, the generated tools have difficulty in solving new problems sharing the same knowledge points

□ **Limited Diversity**

- The complete reliance on the problem itself to build static tools, while neglecting the combination and expansion of these tools. As a result, these tools are restricted in functionality and hard to be applied to different problems

Outline

6



AAAI-25 / IAAI-25 / EAAI-25



- Introduction

- **KTCE Framework**

- Experiments

- Conclusion

Problem Definition



AAAI-25 / IAAI-25 / EAAI-25



7

- To address these limitations, we propose the Knowledge-grounded Tool Creation with Evolution Framework (KTCE). It aims to create reusable and diverse toolsets by grounding tools in domain knowledge and evolving them adaptively.
- Given a task with training dataset $D_{train} = \{(p_i, s_i)\}$, the goal of KTCE is to create a Python function - based toolset T^* with two main objectives are:
 - **Tool Reusability:** Tools should solve classes of problems, measured by the tool calling frequency on D_{test}
 - **Toolset Diversity:** Tools should cover a wide range of tasks, reflected by the proportion of solvable problems in D_{test}

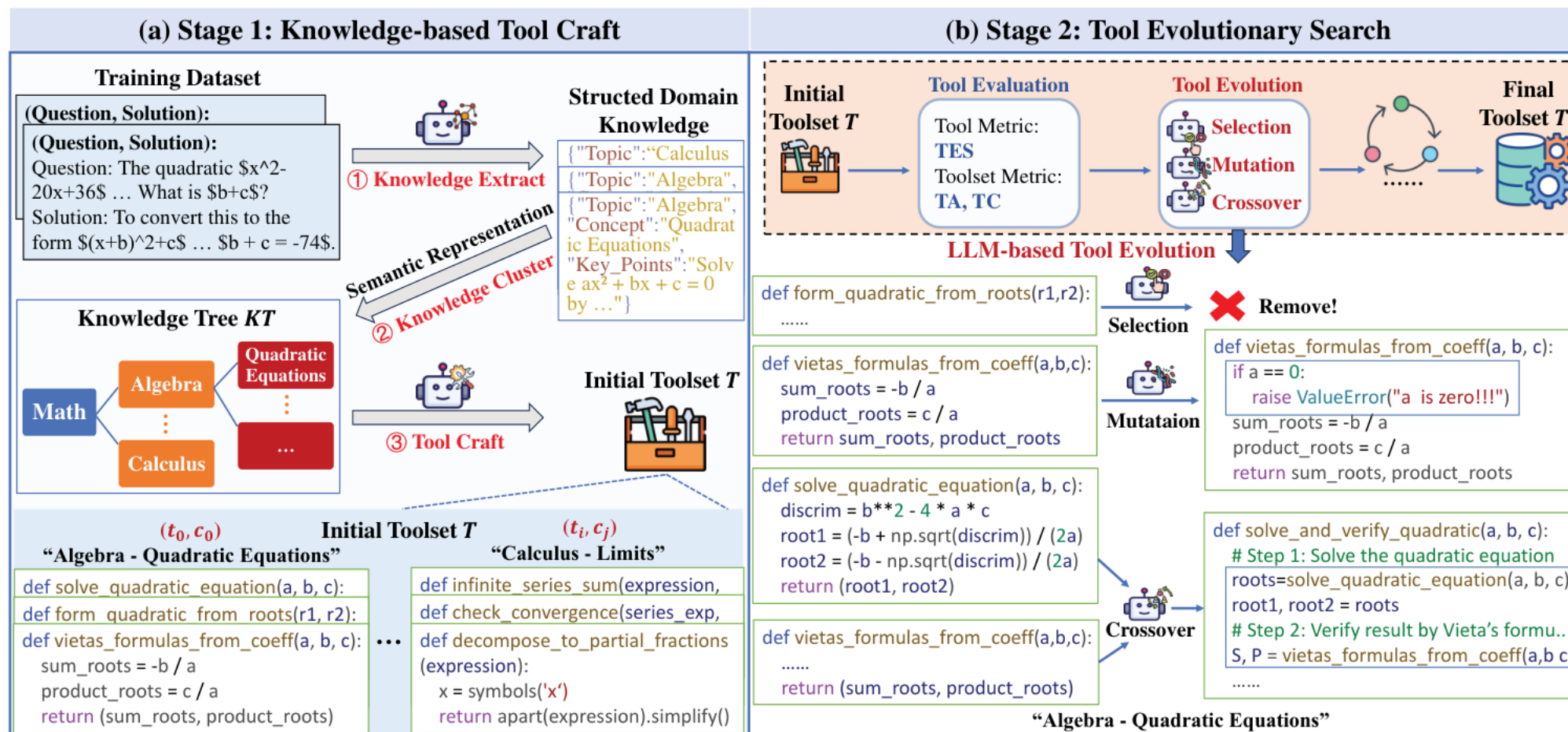


Figure 2: Our framework of KTCE, which consists of two main stages: (a) Stage 1 creates an initial toolset T based on a structured knowledge tree KT , and (b) Stage 2 optimizes the toolset into T^* through iterative tool evaluation and evolution.

□ Stage 2: Tool Evolutionary Search

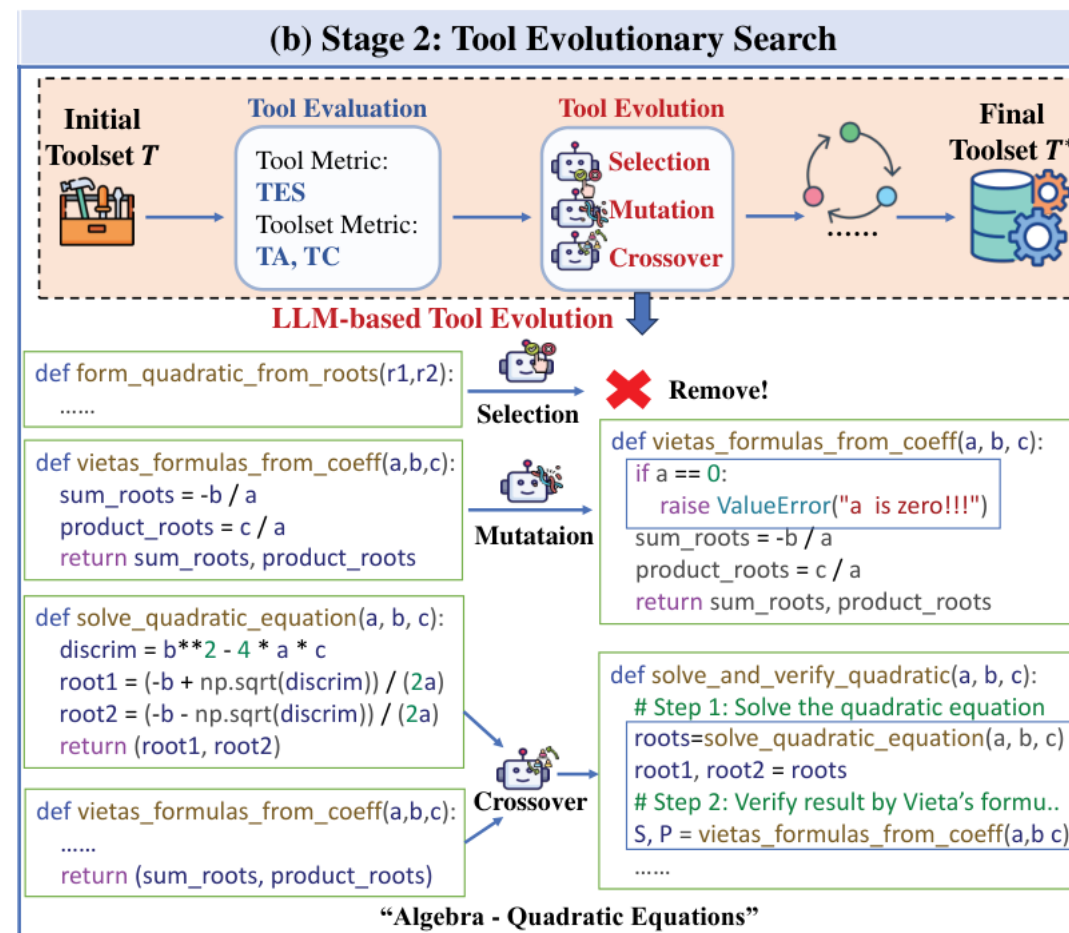
□ Tool Evaluation

- Evaluate toolset using metrics like tool invocation frequency, toolset coverage, task accuracy, along with an optimization function

□ Tool Evolution

- **Selection:** Keep good tools, remove bad ones (e.g., rarely used)
- **Mutation:** Use LLM to refine tools with execution feedback, adjusting code and handling errors
- **Crossover:** Combine tools to make new, more useful functions

□ Iteratively Evaluation & Evolution



□ KTCE-augmented Agent (KA)

□ Utilize tools from the final optimized toolset T^* for generating code solutions of input problems

■ A is the final standardized answer

■ M is the KA's problem-solving process

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

□ Workflow

■ **Tool Retrieval:** Identify relevant (t_i, c_j) pair to access tools from F_{ij} .

■ **Solution Generation:** Leverage LLM to generate Python code calling retrieved tools, incorporating examples from H via In-Context Learning.

■ **Result Formatting:** Execute code, transform intermediate results into standardized answer format A for evaluation



- Introduction
- KTCE Framework
- **Experiments**
- Conclusion

□ Datasets

- Mathematical Reasoning: MATH; Tabular Reasoning: TabMWP; Scientific Reasoning: SCIBENCH

□ Baselines

- Basic: CoT, PoT; Tool-Augmented: PoT with Library (e.g., SymPy), Wolfram
- Tool Creation: Creator, Creator (SR), CRAFT, TROVE

□ Evaluation Metric

- *Acc*: measures model reasoning ability
- *Cov*: task coverage represents the proportion of problems that can be solved by using the tools
- *# Freq*: tool usage frequency represents the average number of times each tool is called
- *# T – size*: the number of functional tools in the toolset



Overall performance

ACC Comparison

- Our KTCE outperforms all baselines across three datasets
- This demonstrates KTCE's ability to handle competition-level reasoning tasks effectively

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	<u>58.80</u>	51.90	<u>33.40</u>	29.90	<u>57.22</u>	68.66	22.16	78.20	30.00
	Wolfram	55.27	37.76	28.60	20.49	34.81	61.77	<u>26.92</u>	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	<u>28.90</u>	48.70	62.34	19.78	<u>87.80</u>	<u>32.00</u>
	CRAFT	53.33	42.62	22.96	25.25	41.67	<u>69.35</u>	20.51	77.00	28.00
	TROVE	57.03	<u>52.00</u>	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 ($p < 0.05$).



□ Overall performance

□ Cov Comparison

- 64.5% on MATH (vs. 14.78% for TROVE), indicate that KTCE tools can solve a significantly broader range of problems.

□ # Freq Comparison

- 3.10 avg. calls/tool (vs. 0.56 for TROVE), demonstrate that KTCE tools are highly reusable across different tasks.

□ # T-size

- Show the larger and more diverse toolset compared to CRAFT

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.

□ Ablation Study

□ “w/o Stage 1”

- Indicate that the structured domain knowledge in Stage 1 is crucial for creating a comprehensive toolset

□ “w/o Stage 2”

- Evolutionary optimization is necessary to ensure toolset quality
- w/o Sel: lead to redundant, ineffective tools
- w/o CO: limite toolset diversity
- w/o Mut: prevent necessary tool updates, reducing adaptation capability

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.

□ Code Complexity Analysis

□ Metrics

- Cyclomatic Complexity (CC)
- Halstead Volume (HV)
- Halstead Effort (HE)

□ Analyse

- KTCE produces solutions with the lowest complexity across all metrics
- Utilizing comprehensive and reusable tools, KTCE enables LLMs to focus on high level reasoning which reduces solution complexity and enhances reasoning efficiency

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.

Reasoning Across Difficulty Levels

- To assess KTCE's robustness across difficulty levels, we analyze its performance on problems of varying difficulty
- KTCE outperforms baselines at all levels with notable improvements on challenging problems
- Validate KTCE's effectiveness in enhancing reasoning capabilities across difficulty levels through the well-designed toolset

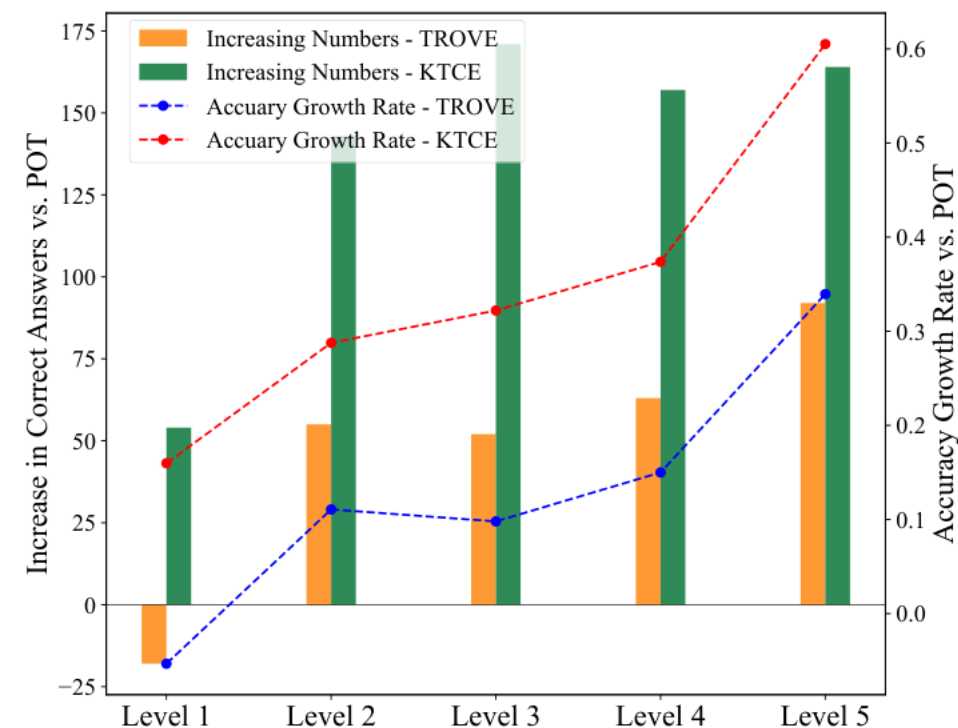


Figure 3: Comparison of Reasoning Accuracy of TROVE and KTCE w.r.t. Difficulty Level in the MATH dataset.



□ Generalization Performance

□ Cross-Dataset

- Toolset generated from MATH dataset applied to GSM8K
- Accuracy improves by 3.34%, showing it captures math concepts for cross - dataset generalization

□ Cross-Model

- GPT-3.5-Turbo's tools used in GPT-4o-Mini on MATH:
4.06% accuracy boost; In DeepSeek-Coder on SCIBENCH:
5.00% accuracy increase
- Demonstrates cross - model robustness



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

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



- Introduction
- KTCE Framework
- Experiments
- **Conclusion**

□ Summary

- Proposed a two-stage framework combining structured knowledge trees (for reusability) and evolutionary search (for diversity), enabling LLMs to:
 -  Induce tools from fundamental concepts: by grounding tools in abstract domain knowledge
 -  Evolve toolsets via Selection/Mutation/Crossover through iterative refinement, KTCE expands toolset diversity
- Demonstrated the effectiveness of KTCE in enhancing LLMs' reasoning capabilities across various tasks.

□ Future works

- Explore more advanced tools such as multi - modal and embodied AI tools
- Design better agent workflows to improve tool selection and utilizing



Thanks for Listening!

zhyma@mail.ustc.edu.cn