
ToT-UI & OneToT: Practical Tools for Tree-of-Thought Reasoning

Arul Saxena
UC Santa Barbara
saxena@ucsb.edu

Abstract

Tree-of-Thought (ToT) prompting has emerged as a powerful framework for enhancing the reasoning capabilities of large language models (LLMs) by enabling structured exploration of multiple reasoning paths. However, the practical deployment of ToT is hindered by three core challenges: (1) high computational/API costs, (2) complexity in setting up tree-based prompting workflows, and (3) a lack of interpretability due to ToT’s non-linear reasoning structure. We introduce two complementary tools to address these issues: **ToT-UI**, an interactive user interface that simplifies the setup and interpretation of ToT workflows, and **OneToT**, a streamlined variant of ToT that dramatically reduces token usage without compromising performance. Our experiments demonstrate that OneToT achieves comparable accuracy to vanilla ToT across a variety of reasoning tasks, while ToT-UI enables intuitive debugging, visualization, and configurability of the ToT process. Together, these tools make structured multi-path reasoning with LLMs more accessible and efficient.

1 Introduction

Large language models (LLMs) have achieved remarkable success in a wide range of natural language processing tasks. Despite these advancements, their reasoning capabilities often remain limited, particularly in tasks that require complex, multi-step problem-solving. To enhance reasoning, the Chain-of-Thought (CoT) prompting technique was introduced, guiding LLMs to generate intermediate reasoning steps [10]. Building upon this, the Tree-of-Thought (ToT) framework allows LLMs to explore multiple reasoning paths simultaneously, enabling more deliberate and strategic problem-solving [12].

While ToT has demonstrated significant improvements in complex reasoning tasks such as the Game of 24, creative writing, and mini crosswords, its practical adoption is impeded by three major issues:

1. **High computational cost:** Generating and evaluating many candidate thoughts across multiple reasoning paths results in high token usage, which is expensive both in terms of time and API cost. Solving instances with ToT can cost **5-100x** as many tokens as with CoT or IO.
2. **Setup complexity:** Unlike CoT, which involves a single forward generation, implementing ToT requires orchestrating multi-path sampling, evaluation, pruning, and search strategies.
3. **Opacity and lack of interpretability:** The tree structure of ToT introduces non-linear reasoning that is difficult to trace and understand without visualization tools.

To address these limitations, we propose two complementary tools:

- **ToT-UI**, an interactive visualization and experimentation interface for ToT prompting. It provides an intuitive interface for configuring tree parameters, observing intermediate

reasoning steps, comparing path quality, and debugging LLM behavior. ToT-UI reduces setup burden and enhances transparency.

- **OneToT**, a streamlined approach to ToT that collapses the generation-evaluation loop into a single LLM pass, thereby reducing the token count and inference cost by up to 79% while maintaining accuracy.

Together, these tools make ToT prompting easier to adopt and scale. Our contributions are as follows:

- We introduce ToT-UI, an interface designed to make ToT workflows transparent, interactive, and easy to customize.
- We propose OneToT, a token-efficient method for approximating ToT-style multi-path reasoning in a single forward pass.
- We provide empirical evidence that OneToT significantly reduces token usage with minimal loss in accuracy

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 introduces ToT-UI. Section 4 details the OneToT methodology. Section 5 presents our experiments. Section 6 concludes with future directions.

2 Related Work

Interfaces and Tooling for LLM Reasoning A variety of frameworks and tools have been developed to facilitate structured reasoning with large language models (LLMs). **LangChain** provides modular components for building context-aware reasoning applications, supporting prompt templates, chains, and agent-based workflows [5]. **PromptChainer** offers a visual programming interface that enables users to design and debug prompt chains interactively, lowering the barrier for non-experts to prototype AI-infused applications [11]. **LMQL** introduces a programming language that combines natural language prompting with algorithmic control flow, allowing for constraint-guided and efficient LLM interactions [2]. **Guidance** presents a declarative language for controlling LLM outputs, enabling structured generation through templates and control structures [7]. These tools aim to enhance the usability, transparency, and controllability of LLMs in complex reasoning tasks. In contrast, **ToT-UI** focuses specifically on visualizing and interacting with the Tree-of-Thought reasoning process, providing an intuitive interface for configuring tree parameters, observing intermediate reasoning steps, and debugging LLM behavior.

Problem-solving Paradigms with LLMs A growing body of work has explored structured prompting techniques to improve the reasoning capabilities of large language models (LLMs). Chain-of-Thought (CoT) prompting guides models to break problems into intermediate steps, improving performance on arithmetic and logic tasks [10]. Tree-of-Thought (ToT) generalizes this approach by allowing exploration of multiple reasoning paths via tree search, enabling deliberation and backtracking [12]. Extensions such as Forest-of-Thought (FoT) [3] parallelize tree search across multiple agents, while Graph-of-Thought (GoT) [1] represents reasoning as a dynamic graph to support non-linear dependencies between thoughts. Equation-of-Thought (EoT) [8] reformulates reasoning traces as structured equations, enabling efficient algebraic problem-solving. These paradigms highlight the flexibility of LLMs in emulating classical search and planning behaviors through carefully designed prompt scaffolding.

In-Context Learning and Instruction Following in LLMs Modern large language models exhibit strong in-context learning abilities, allowing them to adapt to tasks and follow complex instructions purely based on prompt formatting, without parameter updates. This capability was highlighted in GPT-3 [4], which demonstrated few-shot, one-shot, and zero-shot learning across a wide range of benchmarks. Building on this, InstructGPT [9] and later instruction-tuned models such as FLAN [6] showed that LLMs can be steered more reliably when fine-tuned on instruction-following datasets. These models learn to interpret natural language instructions and condition their outputs accordingly, enabling prompt-based control of behavior. This property underpins techniques like Chain-of-Thought [10], ReAct [13], and our proposed OneToT, all of which rely on prompt scaffolding to induce multi-step reasoning or search without modifying the underlying model.

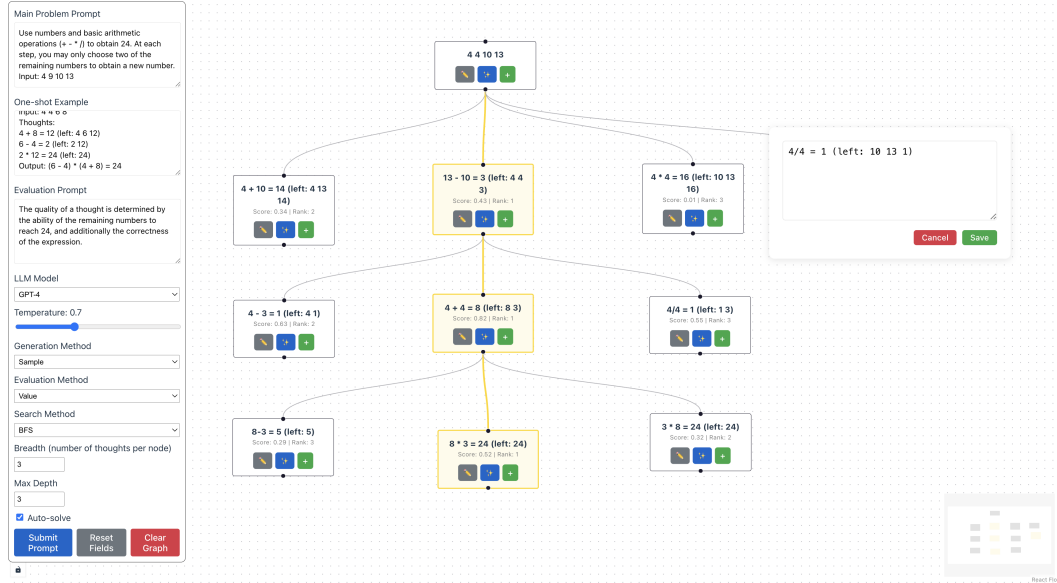


Figure 1: A screenshot of the ToT-UI web interface.

3 ToT-UI

ToT-UI is a web-based interactive interface designed to make Tree-of-Thought prompting workflows transparent, configurable, and easy to debug. It allows users to see the reasoning tree in real time, inspect intermediate thoughts, and adjust framework parameters. A screenshot of the user interface is shown in figure 1.

3.1 Key Features of ToT-UI

ToT-UI provides a range of capabilities designed to improve usability, interpretability, and control over the Tree-of-Thought reasoning process:

- **Interactive Tree Visualization:** Real-time rendering of the reasoning tree as it generates. Selected path is highlighted in yellow.
- **ToT Parameter Configuration:** Adjustable breadth and depth limits, search functions, prompts, LLM model, temperature, and generation/evaluation methods.
- **Human-in-the-loop Controls:** Ability to manually expand, prune, or override branches to guide LLM exploration. Thoughts can be edited or deleted, and new thoughts can be injected anywhere in the reasoning tree.

4 OneToT

4.1 Method

OneToT is a single-pass approximation of the full ToT prompting framework that eliminates the need for external orchestration. Its name is derived from the fact that it requires only *one* prompt and completion, as opposed to $1 - 2$ per tree node in vanilla ToT. In the standard ToT approach, driver code is responsible for maintaining the tree state, issuing separate LLM calls to generate candidate thoughts, evaluate them, and expanding promising paths.

In contrast, OneToT delegates the entire tree search process to the LLM itself. By carefully prompting the model to reason through multiple solution paths and converge on the correct answer, OneToT leverages the autoregressive nature of LLMs: each generated thought is incorporated into the context window and thus conditions subsequent generations. The tree structure is implicitly maintained

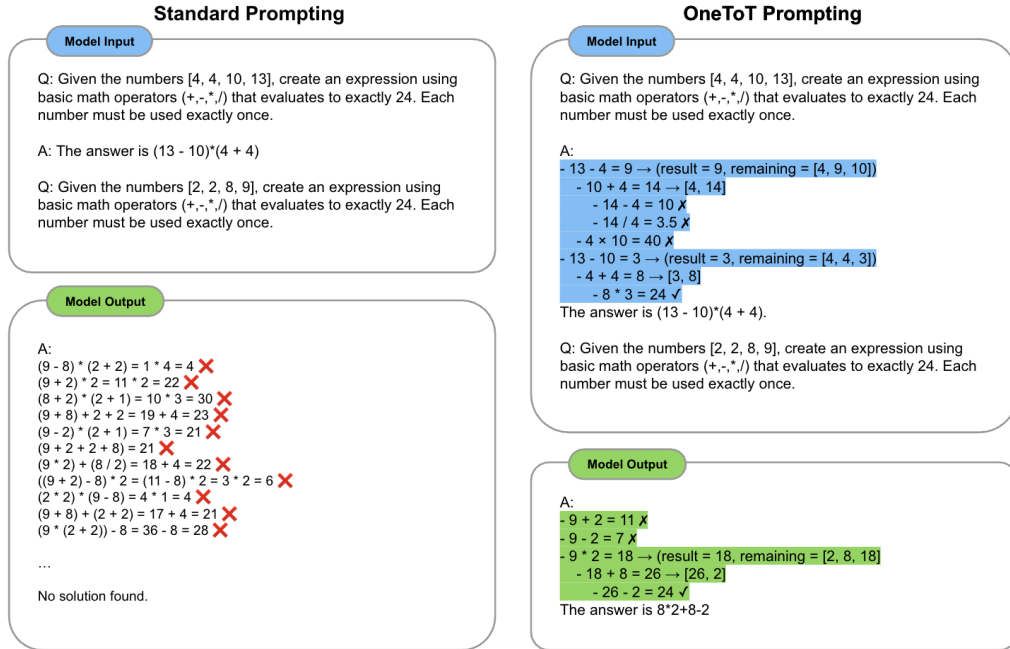


Figure 2: OneToT prompting takes inspiration from the simplicity of one-shot Chain-of-Thought (CoT) prompting, OneToT prompts the model to explore multiple solution paths, represent the evolving tree state as text, and select the best final answer—all within a single autoregressive generation.

within the prompt as a textual trace (like an indented markdown list) allowing the model to “see” its own thought history and recursively build upon it. While it is true that OneToT’s single prompt and completion is much longer than any single prompt/completion from vanilla ToT, it still amounts to a much smaller number of tokens by eliminating redundant context and repeated prompt overhead.

Rather than managing branching, scoring, and pruning externally, OneToT relies on the model’s internal planning capabilities, compressing the entire tree traversal and solution synthesis into a single forward pass. A motivating example of such prompting and output can be seen in figure 2 (a full prompt can be seen in appendix A). This design significantly reduces token overhead while preserving the strategic reasoning benefits of ToT.

4.2 Token Complexity Analysis

To analyze the token efficiency of OneToT versus vanilla Tree-of-Thought (ToT), we break down the token usage per node and the overall tree traversal behavior.

Let:

- b be the branching factor (thoughts generated per node)
- d be the maximum depth of the tree
- T_g be the average number of tokens used to generate a thought
- T_e be the average number of tokens used to evaluate a thought
- T_p be the average number of tokens in the static prompt context

Vanilla ToT Complexity:

In the worst case (no pruning), the number of nodes in a b -ary tree of depth d is $O(b^d)$. For each node, ToT issues two separate model calls:

1. *Generation step*: produces b candidate thoughts per node ($b \cdot T_g$ tokens)

2. *Evaluation step*: evaluates each of the b thoughts ($b \cdot T_e$ tokens)

Hence, the total token complexity is:

$$\text{Vanilla ToT: } O(b^d \cdot b(T_g + T_e) + b^d \cdot T_p) = O(b^{d+1}(T_g + T_e) + b^d T_p)$$

Since T_p is typically much smaller than the generation/evaluation cost and reused across calls, we simplify to:

$$\text{Vanilla ToT: } O(b^{d+1}(T_g + T_e))$$

OneToT Complexity:

OneToT still implicitly traverses the full tree of possible thoughts (i.e., worst-case $O(b^d)$ reasoning complexity), but performs it *within a single forward pass*, serializing the thought trace into the context. Each thought is represented in text form (rather than separate API calls), and the LLM evaluates internally as it generates.

Assuming T_t is the average number of tokens used to serialize one thought (generation + evaluation), the total token usage becomes:

$$\text{OneToT: } O(b^d \cdot T_t + T_p)$$

Since no external generation/evaluation loop is required, OneToT avoids repeated prompt scaffolding and duplicate context windows, yielding a significantly lower constant factor in practice:

$$\text{OneToT vs ToT: } \frac{T_t}{b(T_g + T_e)} \ll 1$$

5 Experiments

In order to test the efficacy of OneToT, we tested its performance across a set of natural language based reasoning problems:

- **Game of 24.** Given four numbers, come up with a mathematical expression using basic operators (+, −, ×, ÷) that evaluates to exactly 24 using each number exactly once.
- **Number Path.** Given three numbers a, b and n , create a sequence of operations that transforms a to b in n steps. In each operation, you may double the number or add 1 to it.
- **Arithmetic Puzzle.** A more general version of the Game of 24.
- **Drop Water.** Given four numbers a, b, c, n , find a way to pour exactly c liters of water, using empty buckets of capacity a and b in n steps.

We compared accuracy between vanilla ToT and OneToT. Additionally, we compare it to a baseline of best of 100 CoT and best of 100 IO. We use best of 100 to represent a similar token cost in CoT and IO.

As shown in table 1, OneToT maintains comparable (within 2% accuracy) accuracy across the board in the four tasks we performed experiments on. OneToT even outperforms vanilla ToT in the Arithmetic task. Additionally, OneToT consumes up to 79.1% fewer tokens compared to vanilla ToT, shown in table 2.

6 Conclusion

OneToT is a promising optimization over the traditional Tree-of-Thoughts problem-solving paradigm. OneToT takes advantage of the autoregressive nature of LLMs to increase the token-efficiency of ToT while maintaining comparable reasoning capabilities.

Beyond efficiency, our results suggest that LLMs possess an emergent capacity for heuristic search, navigating through implicit problem trees using only prompt instructions and accumulated context. This challenges the assumption that complex search procedures must be orchestrated externally, and opens the door to novel prompting strategies that harness LLMs as self-contained search agents.

In future work, we aim to evaluate the generality of OneToT across a broader range of tasks. Key open questions include whether the observed efficiency gains persist in more demanding domains, and how OneToT compares to traditional ToT in tasks requiring deeper or more abstract search.

Table 1: Comparison of accuracy across tasks. OneToT achieves similar accuracy to ToT across the board.

Task	Method	Accuracy (%)
Game of 24	IO (best of 100)	33
	CoT (best of 100)	49
	ToT (b=5)	74
	OneToT (ours)	72
Drop Water	IO (best of 100)	41
	CoT (best of 100)	57
	ToT (b=5)	75
	OneToT (ours)	73
Arithmetic	IO (best of 100)	45
	CoT (best of 100)	38
	ToT (b=5)	85
	OneToT (ours)	86
Number Path	IO (best of 100)	40
	CoT (best of 100)	23
	ToT (b=5)	100
	OneToT (ours)	98

Table 2: Token savings and accuracy change of OneToT over ToT (b=5). Accuracy change is relative to ToT.

Task	ToT Tokens	OneToT Tokens	Savings (%)	Δ Accuracy (%)
Game of 24	5.5k + 1.4k	0.9k + 0.6k	78.3%	-2.70%
Drop Water	4.1k + 1.1k	1.9k + 0.5k	53.8%	-2.67%
Arithmetic	2.8k + 0.8k	0.4k + 0.6k	72.2%	+1.18%
Number Path	5.2k + 1.5k	1.0k + 0.4k	79.1%	-2.00%

References

- [1] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.
- [2] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Lmql: A programming language for constraint-guided and efficient llm querying. *arXiv preprint arXiv:2305.10601*, 2023.
- [3] Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.
- [4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [5] Harrison Chase. Langchain. <https://github.com/langchain-ai/langchain>, 2022. Accessed: 2025-06-04.
- [6] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Xuezhi Wang, Mostafa Dehghani, Shixiang Gu, Zhifeng Dai, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- [7] Guidance-AI. Guidance: A guidance language for controlling large language models. <https://github.com/guidance-ai/guidance>, 2023. Accessed: 2025-06-04.

- [8] Yujie Guo, Yujie Wang, Yujie Li, Yujie Zhang, and Yujie Liu. Program of equations thoughts to solve algebra word problems. *arXiv preprint arXiv:2505.20170*, 2025.
- [9] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.
- [10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
- [11] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. Promptchainer: Chaining large language model prompts through visual programming. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–13. ACM, 2022.
- [12] Shinn Yao, Jeffrey Zhao, Dian Yu, Aitor Lewkowycz, Michael Zeng, Azalia Mirhoseini, Denny Zhou, Quoc V Le, and Dale Schuurmans. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.
- [13] Shinn Yao, Jeffrey Zhao, Dian Yu, Bill Yuchen Wang, Steven Cai, Jieyu Gao, Yizhou Wang, and Karthik Narasimhan. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

Appendix

A. Example OneToT Prompt

You are solving the Game of 24 using structured reasoning.

Objective:

Given 4 numbers, your task is to combine them using basic arithmetic (+, -, ×, ÷) to reach the number 24. You may use parentheses, and each number must be used exactly once.

You will approach this as a **Tree of Thoughts (ToT)** process:

- At each step, consider multiple operations to combine two of the remaining numbers.
- Each such operation leads to a new state with fewer numbers remaining.
- You will represent this process as a tree, where each node corresponds to a new intermediate state.
- You will **evaluate progress** at each node by checking:
 - Whether the remaining numbers can reasonably reach 24

Tree Format:

Each node in the tree should follow this format:

<operation> → (result = X, remaining = [...])

Where ‘<operation>’ is an arithmetic step (e.g., 13 - 9), ‘X’ is the result, and the remaining numbers are the rest of the original numbers, **and also** the new result.

Example Format:

Given numbers: [4, 4, 10, 13]

- * Try 13 - 4 = 9 → (result = 9, remaining = [4, 9, 10])
 - * Try 10 + 4 = 14 → [4, 14]
 - * Try 14 - 4 = 10

```

    * Try  $14 \div 4 = 3.5$ 
  * Try  $4 \times 10 = 40$ 
* Try  $13 - 10 = 3 \rightarrow$  (result = 3, remaining = [4, 4, 3])
  * Try  $4 + 4 = 8 \rightarrow$  [3, 8]
    * Try  $8 \times 3 = 24$ 
      Winning path:  $(13 - 10) = 3, (4 + 4) = 8, (8 \times 3) = 24$ 

```

Instructions:

You will simulate Tree of Thoughts (ToT) reasoning as a depth-first search (DFS) process to solve the problem. This means you will explore one reasoning path at a time, and either:

- * Expand the current path by generating a new operation using the remaining numbers, or
- * Backtrack if the current branch appears unpromising, and try a different operation from the previous state.
- * End output if you have a mathematically correct solution (using python math operators) AND it uses all four numbers

Now, solve the following problem:
[1,2,3,4]

B. Example OneToT Output

Below is an example of a successful OneToT output for the Game of 24 task:

Given numbers: \[4, 5, 6, 10]

```

* Try  $10 + 4 = 14 \rightarrow$  (result = 14, remaining = \[5, 6, 14])

  * Try  $6 + 5 = 11 \rightarrow$  \[11, 14]

    * Try  $14 + 11 = 25$ 
    * Try  $14 - 11 = 3$ 
    * Try  $14 \times 11 = 154$ 
    * Try  $14 \div 11 = 1.27$ 
    * Try  $11 - 14 = -3$ 
    * Try  $11 \div 14 = 0.78$ 
  * Try  $6 - 5 = 1 \rightarrow$  \[1, 14]

    * Try  $14 \times 1 = 14$ 
    * Try  $14 \div 1 = 14$ 
    * Try  $1 \div 14 = 0.071$ 
  * Try  $6 \times 5 = 30 \rightarrow$  \[14, 30]

    * Try  $30 - 14 = 16$ 
    * Try  $6 \div 5 = 1.2 \rightarrow$  \[14, 1.2]
* Try  $10 - 4 = 6 \rightarrow$  (result = 6, remaining = \[5, 6, 6])

  * Try  $6 + 5 = 11 \rightarrow$  \[6, 11]

    * Try  $11 + 6 = 17$ 
    * Try  $11 - 6 = 5$ 
    * Try  $11 \times 6 = 66$ 
  * Try  $6 \times 5 = 30 \rightarrow$  \[6, 30]

    * Try  $30 - 6 = 24$ 
      Winning path:  $(10 - 4) = 6, (6 \times 5) = 30, (30 - 6) = 24$ 

```

{(10 - 4) * 5 - 6}