# TSLM: Tree-Structured Language Modeling FOR DIVERGENT THINKING

# Anonymous authors

000

001

002 003 004

006

008 009

010 011

012

013

014

015

016

017

018

019

021

023

024

025

026

027

028 029

031

033

034

037

040

041

042

043

044

046

047

048

051

052

Paper under double-blind review

# **ABSTRACT**

Current language models generate solutions through sequential reasoning, limiting their ability to systematically explore multiple solution paths. We introduce Tree-Structured Language Modeling (TSLM), which teaches language models to generate complete search trees within a single generation process using special tokens to encode branching structure. TSLM serializes tree exploration into linear sequences, enabling standard transformer training on tree-structured reasoning traces that capture both successful and failed solution attempts. Across structured planning (Game of 24, Gridworld) and open-ended reasoning tasks (ProntoQA, GSM8K), TSLM achieves superior performance: 100% accuracy on Game of 24 vs. 17% for sequential baselines, and robust extrapolation to 20×20 grids (76.5%) compared to Tree-of-Thought's collapse (26%). Remarkably, TSLM demonstrates 14× parameter efficiency, with a 0.5B model (68% scaling performance) outperforming 7B sequential baselines (19-26%). TSLM also exhibits emergent capabilities including unsolvable problem detection and rapid adaptation with minimal training data. These results challenge the assumption that reinforcement learning is necessary for robust reasoning, demonstrating that supervised learning on complete tree-structured traces provides an efficient alternative for developing systematic exploration capabilities in language models.

#### Introduction

Complex reasoning often requires exploring multiple solution paths before converging on an answer. Consider solving the Game of 24 with numbers [8, 4, 3, 6]: a systematic approach would explore  $(8+4) \times (6 \div 3) = 24$ ,  $(8-4) \times 6 \div 3 = 4$ , and other combinations simultaneously, rather than committing to a single path early. However, current language models generate solutions sequentially, making it difficult to systematically explore alternatives or recover from early mistakes.

Recent reasoning models like o1 OpenAI (2024) and DeepSeek-R1 DeepSeek-AI et al. (2025) have shown impressive capabilities through extended reasoning traces, but still fundamentally operate as sequential generators. While they may internally consider multiple options, they cannot explicitly represent parallel exploration within their generation process. External methods like Tree-of-Thought Yao et al. (2023) address this through post-hoc search, but require multiple independent model calls and external orchestration.



deterministic path.

(a) Sequential thinking: (b) Tree-structured think-A purely sequential ap- ing: A branching exploproach that traverses one ration that expands multiple possibilities simultaneously.

Figure 1: Sequential vs. Tree-Structured Reasoning. (a) Sequential approaches commit to single paths, limiting exploration of alternatives. (b) Tree-structured approaches systematically explore multiple possibilities, enabling recovery from mistakes and comprehensive solution space coverage. TSLM bridges this gap by teaching language models to generate tree-structured explorations natively.

To address these limitations, we introduce Tree-

Structured Language Modeling (TSLM), which directly embeds systematic exploration into language generation. Rather than forcing tree-structured reasoning into sequential format or applying search externally, TSLM teaches models to natively generate complete search trees. Using special tokens to encode branching structure ([SEP1] for viable paths, [SEP2] for dead ends, [SEP3] for step completion), TSLM serializes tree exploration into trainable sequences while preserving the parallel exploration capability.

This approach enables models to learn the complete reasoning process—including both successful paths and failed attempts—leading to more robust problem-solving capabilities. Our experimental results demonstrate that this internalized search strategy not only outperforms sequential baselines but also exhibits remarkable parameter efficiency and emergent capabilities that arise naturally from learning complete exploration patterns rather than just final answers.

# 1.1 OUR CONTRIBUTIONS

This paper makes the following contributions:

- We introduce Tree-Structured Language Modeling (TSLM), a token-based serialization framework that enables standard transformers to learn tree-structured reasoning through supervised learning on complete search traces
- We demonstrate consistent performance gains across diverse tasks: 100% vs. 17% on Game of 24, robust extrapolation to larger Gridworld environments (76.5% vs. 26% for Tree-of-Thought), and competitive performance on open-ended reasoning
- We reveal remarkable parameter efficiency, showing that a 0.5B TSLM model (68% scaling performance) outperforms 7B sequential and RL baselines (19-26%), representing a 14× parameter efficiency advantage
- We uncover emergent capabilities including systematic identification of unsolvable problems, rapid adaptation with minimal training data (95% accuracy after 1.5K samples), and robust extrapolation beyond training complexity
- We provide detailed analysis of inference-time scaling dynamics, showing TSLM's internalized search procedures generalize better than external search algorithms when complexity exceeds training boundaries

Our findings challenge the prevailing assumption that reinforcement learning is necessary for developing robust reasoning capabilities in language models. Instead, we demonstrate that properly structured supervised learning of tree-formatted reasoning traces may provide a more direct and efficient path toward enhanced reasoning and planning. We provide detailed answers to common questions in Appendix §A.

# 2 BACKGROUND

Contemporary language models generate tokens sequentially, modeling  $p(y \mid x) = \prod_{t=1}^{|y|} p(y_t \mid x, y_{< t})$  Brown et al. (2020). For problems requiring exploration of multiple solution paths, this sequential approach has limitations: (1) linear commitment to single paths, (2) error propagation, (3) redundant computation when multiple solutions are needed, and (4) inability to systematically explore alternatives in parallel.

Current multi-path reasoning approaches rely on post hoc sampling methods. Tree-of-Thoughts Yao et al. (2023) samples multiple candidates at each step using external search algorithms, but faces exponential computational costs. Recent autoregressive models like of OpenAI (2024) generate reasoning procedures in single sequences, but remain constrained by linear generation and may produce redundant information Chen et al. (2025). We provide a more extensive related works in Appendix §B.

# TREE-STRUCTURED LANGUAGE MODELING (TSLM)

We introduce Tree-Structured Language Modeling (TSLM) as a framework to natively incorporate divergent exploration for natural language generation. TSLM differs from standard sequential language modeling by generating multiple possible next actions or statements and linking them into a coherent tree structure.

113 114 115

116

117

118

119

120

121

122

108

109 110

111

112

### 3.1 Modeling Multiple Next Actions

Let s be the current state or partial solution. In a sequential language model, we predict a single next action a from s and transition to s' = T(s, a). By contrast, in TSLM, we represent multiple possible successors:

$$\pi_{\theta}(s) = \{T(s, a_1), \cdots, T(s, a_k)\},\$$

where each  $a_i$  denotes a distinct branch and k is the branching factor. TSLM learns to expand s into these k successors within a single forward pass, retaining the relationships among them rather than generating them independently.

123 124 125

### 3.2 Encoding and Decoding with Tree Structure

126 127 128

To enable standard transformer architectures to learn tree-structured reasoning, we develop a serialization scheme that encodes complete search trees into linear sequences. This approach allows us to train language models on tree data while preserving the branching structure.

129

### **Token-Based Tree Serialization.** We introduce special tokens to encode tree structure:

130 131

• [SEP1]: Indicates a viable action that can be further expanded

132 133

• [SEP2]: Indicates a non-viable action (dead end)

134 135

• [SEP3]: Marks the end of all actions at the current step • [BOS] and [EOS]: Mark sequence boundaries

136 137

138

This serialization captures both successful paths and unsuccessful explorations, teaching the model the complete search process rather than just final answers. A detailed worked example showing the complete serialization format is provided in Appendix §C.

139 140 141

**Training Procedure.** During training, we apply standard language modeling loss to the entire serialized sequence:

142 143 144

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(y_t | y_{< t}, x) \tag{1}$$

145 146

where  $y_t$  includes both reasoning content and structural tokens. Crucially, this trains the model to:

147

1. Generate multiple actions at each decision point

148 149

2. Assign appropriate viability markers ([SEP1] vs [SEP2]) 3. Structure the exploration systematically

3. Maintain a queue of unexplored viable states

150 151

**Inference Procedure.** During inference, TSLM reconstructs the tree structure:

152

1. Generate the next reasoning step with multiple candidate actions

153 154

2. Parse structural tokens to identify viable branches ([SEP1])

156 157

4. Recursively expand each viable state until finding a solution or exhausting options

158 159

161

This approach enables systematic exploration while maintaining computational efficiency through shared computation of common prefixes. Unlike Tree-of-Thought which requires multiple independent forward passes, TSLM generates the complete exploration strategy in a single coherent

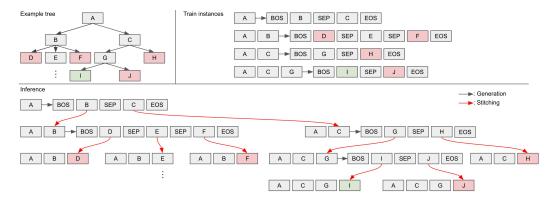


Figure 2: TSLM (Tree-Structured Language Model) is a language model designed for hierarchical exploration in sequence generation tasks. Beginning with an initial state (e.g., "A") and progressing toward a goal state (e.g., "I"), TSLM constructs a tree structure where nodes represent states and branches signify possible paths. During training, the model serializes the tree into linear sequences using special tokens to separate branches and mark the start/end of a sequence, allowing it to learn structured expansions effectively. During inference, TSLM generates multiple branching actions to explore diverse sequences (e.g., expanding from "C" to "G" and "H"). These branches are independently expanded in parallel using a stitching process, enabling broad exploration toward the goal while efficiently pruning unwanted paths.

### 3.3 TRAINING OBJECTIVES AND COMPUTATIONAL CONSIDERATIONS

TSLM training differs from standard sequence modeling in several key aspects that enable effective tree structure learning.

**Tree-Aware Loss Function.** While we apply standard language modeling loss (Equation 2), the training targets include complete tree structures rather than single solution paths. For a tree  $\Gamma(t)$  with N nodes, each node  $s_i$  contributes to the loss:

$$\mathcal{L} = -\frac{1}{N} \sum_{s_i \in \Gamma(t)} \sum_{t=1}^{|y_{s_i}|} \log p(y_{s_i,t}|y_{s_i,< t}, x)$$
 (2)

where  $y_{s_i}$  is the token sequence for node  $s_i$ . This ensures the model learns both successful and unsuccessful exploration patterns.

**Structural Consistency Training.** Beyond token prediction, TSLM must learn to generate structurally valid trees. We achieve this through:

- Marker consistency: Training on diverse tree structures teaches proper [SEP1]/[SEP2] usage
- **Branching patterns**: Models learn appropriate branching factors for different problem types
- Termination conditions: Proper [SEP3] placement indicates complete step exploration

**Computational Efficiency.** TSLM training requires  $O(N \cdot L)$  computation where N is the average number of nodes per tree and L is the average sequence length per node. While this is more expensive than training on single paths (O(L)), it provides richer supervision that leads to better generalization with fewer training examples.

# 4 SEARCH TREE SUPERVISION

#### 4.1 Training on Structured Tasks

For structured tasks with predefined search trees, we can directly train TSLM to learn and reproduce the tree structure. Formally, let  $t \in \mathcal{T}$  be a task with a solution of a finite depth, T(s,a) be a transition function that maps state-action pairs to new states, and A(s) be a finite action space that defines valid actions at state s. Since these components are explicitly defined, we can generate the complete search tree and employ TSLM to predict the branching structure  $(T(s,a) \mid a \in A(s))$  at each state in the search tree. By directly imitating the predefined tree expansions, TSLM guides the model to faithfully reproduce structured exploration patterns. Examples include board games and planning problems with well-defined rules.

### 4.2 SEARCH TREE SUPERVISION FOR OPEN-ENDED REASONING TASKS

While structured tasks have predefined search trees that TSLM can directly learn from, most real-world tasks lack explicit tree structures, providing only correct answers or gold trajectories. We introduce a bootstrapping method to construct synthetic training trees by combining model-generated explorations with known solutions.

For each training instance, we employ a supervision language model to generate candidate actions via Tree-of-Thoughts sampling Yao et al. (2023). The process involves:

- 1. Sampling a set of candidate actions at each state using beam search
- 2. Building a tree structure by propagating these actions forward
- 3. Incorporating known gold trajectories as high-priority branches
- 4. Ordering remaining branches using a reward function R(s,a)
- 5. Deduplicating redundant paths while preserving the tree structure

Algorithm 1 details this procedure. Our approach ensures each training tree contains at least one valid solution while exploring diverse alternatives. The reward-based ordering helps prioritize promising actions, while deduplication prevents redundant search. We adopt the original RAPHao et al. (2023) reward function to refine exploration by prioritizing promising branches.

### 5 EXPERIMENTAL RESULTS

### 5.1 Baselines for Comparison

**Baseline Model Architecture** We compare TSLM to the following baselines:

- Sequence Cloning (SC): A standard sequential modeling that clones a single linear sequence of gold Chain-of-Thought (language modeling similar to GPT-3 (Brown et al., 2020) or Llama 3 (Touvron et al., 2023)).
- **Procedure Cloning (PC)**: A sequential modeling that clones Chain-of-Thought reasonings of the entire search trace in a single linear sequence (o1-like reasoning models, trained with systematic supervision(Kim et al., 2024; Yang et al., 2022)).
- **GRPO**: A reinforcement learning approach to incentivize reasoning trace during post-training (o1-like reasoning models, trained with GRPO (Shao et al., 2024) objectives)
- Tree-of-Thought (ToT): Scaling the number of inference of SC model during test-time using external search algorithms (Yao et al., 2023) with beam search across multiple reasoning paths.

We test models using greedy decoding for sequential methods. For TSLM inference, we explore different tree traversal and solution selection strategies. Unless otherwise specified, we use Breadth-First Search (BFS) as the default algorithm to systematically explore the generated tree structure until finding a successful solution. (We analyze implications of BFS versus alternative search strategies

# Algorithm 1 Guided Search Tree Bootstrapping

270

271

272

273

274

275

276

277

278

279

281

284

287

289

291

292

293

305

306307308

310

311

312

313

314

315

316

317

318 319 320

321

322

323

```
Input: Task \mathcal{T}, transition T, reward R, branch factor k, supervision model \pi_{\theta}
Data = [
for t \in \mathcal{T} do
  Initialize queue = [s].
  Gold trajectory s_0 = s, s_1, \ldots, s_n = g
  while g \notin \text{queue do}
     tmp = queue.pop(0)
     for i = 1 to k do
        tmpqueue = []
        if i = 1 and tmp \in \{s_0, \ldots, s_n\} then
          a_i = a^*(tmp) {Add gold action}
          tmpqueue.add(T(tmp, a_i))
        else
          a_i \sim \pi_{\theta}(tmp)
          if a_i \notin \{a_1, ..., a_{i-1}\} then
             tmpqueue.add(T(tmp, a_i))
              {Deduplication}
          end if
        end if
     end for
     tmpqueue = \sigma_R(tmpqueue){Sort by reward}
     queue+ = tmpqueue
  end while
  Data.append(queue)
end for
```

	Task	SC	PC	GRPO	ToT	TSLM
§H.2	Game of 24	17.0%	47.0%	15.0%	17.0%	100 %
	Gridworld (10×10)	78.2%	99.7%	24.0%	95.0%	100 %
	Gridworld (20×20)	19.0%	26.5%	6.0%	26.0%	76.5 %
§H.3	ProntoQA	99.7%	97.5%	99.8%	100%	100%
	GSM8K	55.8%	55.9%	60.8%	85.0%	61.6%

Table 1: Success rates across different tasks and methods. The Gridworld results show both in-domain  $(10\times10)$  and scaling  $(20\times20)$  performance, highlighting ToT's dramatic degradation when complexity scales beyond training boundaries.

such as Depth-First Search (DFS) in Appendix  $\S G.2.$ ) For each expansion during the inference, we select the first k=5 actions generated and deduplicate them with exact matching.

To evaluate Tree-Structured Language Modeling (TSLM), we conduct experiments on both structured and unstructured tasks. Our experiments use Llama-3-8B Grattafiori et al. (2024) as base experiments, comparing TSLM against sequential language modeling baselines. Also, We aim to compare architectural differences rather than scaling effects, using modest training data (less than 10K instances per task) with supervised fine-tuning. We have two task scenarios: Structured Planning Tasks, which are tasks with predefined search trees, while Open-ended Reasoning Tasks are tasks with undefined solution spaces. Structured Planning Tasks include Game of 24 and Textualized Gridworld, while Open-ended Reasoning Tasks include ProntoQA and GSM8K. Refer to Appendix §H for more details regarding each task and supervision examples.

### 5.2 Base Results

Table 1 summarizes our experimental findings across all tasks, revealing key insights about different reasoning approaches. TSLM consistently outperforms sequential models, achieving perfect accuracy on structured tasks. Most notably, the Gridworld scaling results reveal a striking limitation of Tree-of-

Thought: while ToT achieves excellent in-domain performance (95.0% on 10×10 grids), it suffers catastrophic degradation when complexity scales (dropping to 26.0% on 20×20 grids).

This ToT scaling failure is particularly surprising given its strong in-domain performance and sophisticated external search mechanisms. In contrast, TSLM maintains robust performance across the complexity boundary ( $100\% \rightarrow 76.5\%$ ), demonstrating that internalized search procedures generalize better than external search algorithms. For open-ended tasks, ToT shows strong performance on GSM8K (85.0%) but this advantage disappears when systematic exploration is needed, as evidenced by Game of 24 results where ToT performs no better than basic sequential methods (17.0%).

### 5.3 PARAMETER EFFICIENCY AND RAPID ADAPTATION

We evaluate TSLM's efficiency along two critical dimensions: parameter scaling and learning speed. These analyses reveal remarkable efficiency characteristics that distinguish TSLM from conventional approaches.

**Parameter Efficiency Analysis.** TSLM demonstrates remarkable parameter efficiency, with a 0.5B model achieving 68% scaling performance while 7B baseline models achieve only 19-26%—representing a 14× parameter efficiency advantage. The scaling curve for TSLM is notably flat, suggesting systematic exploration supervision captures algorithmic competencies largely independent of parameter count. Detailed analysis across the Qwen 2.5 model family is provided in Appendix §F.

**Rapid Adaptation Capability.** TSLM also demonstrates superior learning efficiency on opendomain tasks. As shown in Figure 3, TSLM achieves rapid convergence on ProntoQA, reaching 77.3% and 95.1% accuracy after just 750 and 1500 training samples respectively. Meanwhile, SC and PC require significantly more data to reach comparable performance. Notably, GRPO shows poor initial performance (slightly worse than SFT at the start) before eventually reaching 100% accuracy, highlighting the cold-start problems inherent in RL-based approaches.

### 6 KEY ANALYSIS

Beyond core performance gains, TSLM exhibits several remarkable capabilities that distinguish it from traditional approaches.

# 6.1 IDENTIFYING UNSOLVABLE CASES

One challenging aspect for language models is avoiding hallucination on problems that have no valid solution. Sequential models like SC or PC implicitly learn to generate answers within their training distribution, which can be problematic when faced with unsolvable cases. For example, in the Game of 24 task, the numbers 1, 1, 2, and 3 cannot generate 24 through any sequence of arithmetic operations. Since sequential models have not been trained on examples with no solution, they tend to hallucinate and generate invalid answers.

TSLM, however, demonstrates a unique capability to identify unsolvable cases. We qualitatively analyzed its behavior on the 1, 1, 2, 3 instance and found that TSLM correctly terminated without generating any answer. This suggests the tree-structured exploration enables TSLM to systematically explore the full solution space and recognize when no valid path exists. Unlike sequential models that are pressured to always generate some answer, TSLM's broader search allows it to confidently determine and declare when a problem is unsolvable.

This capability emerges naturally from TSLM's training on complete search trees rather than individual solution paths. By learning to represent both successful and unsuccessful branches, TSLM develops a more complete model of the solution space that includes recognizing unsolvable cases.

### 6.2 Extrapolation beyond Training Data

TSLM demonstrates strong extrapolation capabilities when tested on larger grid sizes than seen during training. Testing on 20×20 grids after training on 10×10 grids, TSLM maintains 91.5%

SC	PC	TSLM (ours)		
33.0%	81.1%	91.5%		

Table 2: Qualitative comparison of model extrapolation capabilities on Gridworlds of varying sizes (maximum  $20 \times 20$ ). Each heatmap shows performance at different grid dimensions (x,y), with darker colors indicating better performance. The red box indicates the boundary of training data. Overall accuracy shown above each plot.

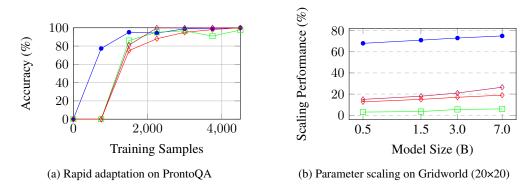


Figure 3: TSLM efficiency analysis. (Left) Rapid adaptation showing GRPO's cold-start problems. (Right) Parameter scaling on Gridworld complexity extrapolation, where 0.5B TSLM (68%) dramatically outperforms 7B baselines, demonstrating 14× parameter efficiency. Methods: TSLM (●), PC (□), SC (△), GRPO (⋄)

accuracy compared to PC's 81.1% and SC's 33.0% (Table 2). This suggests TSLM's structured tree representation learns generalizable navigation patterns rather than memorizing specific configurations.

#### 6.3 RAPID ADAPTATION

These results demonstrate two key efficiency advantages of TSLM. First, the rapid adaptation analysis reveals fundamental differences in learning dynamics: while TSLM shows consistent improvement from the start, GRPO exhibits initial performance that is slightly worse than basic supervised learning before eventually converging to 100% accuracy. This highlights the cold-start brittleness inherent in RL-based approaches when base competence is low.

Second, the parameter scaling results on Gridworld extrapolation tasks reveal TSLM's most remarkable characteristic: algorithmic competencies that are largely independent of parameter count. The flat scaling curve ( $68\% \rightarrow 75\%$  from 0.5B to 7B) contrasts sharply with baseline methods that show steep parameter dependence, suggesting TSLM teaches structured reasoning procedures rather than relying on brute-force memorization. We provide detailed inference-time scaling analysis including performance scaling and base model comparisons in Appendix §G.

#### 6.4 Inference-time Scaling Properties

A key finding is that TSLM's internalized search procedures generalize better than external search methods when complexity exceeds training boundaries. In Gridworld experiments, Tree-of-Thought achieves 95% success on 10×10 training grids but collapses to 26% on 20×20 test grids, while TSLM maintains robust performance (76.5%). This demonstrates that training-time acquisition of search algorithms provides more robust generalization than inference-time application of external

procedures. Detailed analysis of inference-time scaling dynamics, including BFS vs DFS trade-offs and performance on GSM8K, is provided in Appendix §G.

### 7 DISCUSSION

# 7.1 Supervised Learning vs. Reinforcement Learning for Reasoning

Our results challenge a growing trend in the field that emphasizes reinforcement learning for improving reasoning capabilities in language models. Recent models like DeepSeek Math, O1, and Phi-4 have demonstrated enhanced reasoning through extensive RL training, leading to a prevailing belief that RL is necessary for developing robust reasoning.

Our findings suggest an alternative perspective: properly structured supervised learning of complete tree-structured reasoning traces can achieve similar or better performance without the complexity and computational demands of RL. This challenges the "SFT Memorizes, RL Generalizes" paradigm Chu et al. (2025), but offers a crucial nuance—when the supervised data contains the full exploration process, not just input-output pairs, SFT can also lead to strong generalization.

The key insight appears to be whether the model learns the underlying algorithm or merely the answer. By explicitly teaching the algorithm through complete traces, we enable the model to apply the same reasoning process to more complex instances of the same problem class.

# 7.2 COMPARISON WITH ALGORITHMIC IMITATION

TSLM differs from algorithmic imitation approaches that embed reasoning into linear traces. Unlike single-path imitation, TSLM represents multiple branching paths simultaneously and enables dynamic exploration adjustment. Detailed comparison including representation differences, exploration capabilities, and adaptability is provided in Appendix §E.

### 7.3 LIMITATIONS AND FUTURE WORK

TSLM faces computational overhead due to recomputing shared nodes during tree-based training, and synthetic tree generation quality depends heavily on the supervision model's capabilities. Current transformer architectures are not optimized for tree-structured generation, and broader application to complex domains like code generation requires further exploration. Detailed discussion and potential solutions are provided in Appendix §D.

# 8 Conclusion

We introduce Tree-Structured Language Modeling (TSLM), which enables language models to generate complete search trees within a single generation process using token-based serialization. TSLM achieves superior performance across structured planning and open-ended reasoning tasks: 100% accuracy on Game of 24 (vs. 17% for baselines), robust extrapolation to larger environments (76.5% vs. 26% for Tree-of-Thought), and remarkable 14× parameter efficiency where a 0.5B model outperforms 7B baselines.

TSLM demonstrates that supervised learning on tree-structured traces can match or exceed reinforcement learning approaches without their complexity. Key emergent capabilities include identifying unsolvable problems, rapid adaptation with minimal data, and systematic exploration that generalizes better than external search algorithms. The approach requires only standard language model fine-tuning with special tokens to encode tree structure, making it compatible with existing transformer architectures.

These results challenge prevailing assumptions about sequential generation and reinforcement learning necessity for reasoning, demonstrating that properly structured supervised learning provides an efficient path toward enhanced reasoning capabilities in language models.

# REFERENCES

486

487

488

489

490

491

492 493

494

495

496

497

498

499

500 501

502

504

505

506 507

509

510

511

512

513

514

515

516

517

519

521

522

523

524

525

527

528

529

530

531

532

534

535

536

538

- 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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10. 1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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.
- Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2025. URL https://arxiv.org/abs/2412.21187.
- Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training, 2025. URL https://arxiv.org/abs/2501.17161.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.
- Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2024. URL https://arxiv.org/abs/2309.17179.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev,

541

542

543

544

546

547

548

549

550

551

552

553

554

558

559

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

588

590

592

Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai

596

597

598

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621 622

623

624

625 626

627

628

629

630

631

632

633

634

635

636 637

638

639

640 641

642

643

644

645

646

647

Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manay Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

Doyoung Kim, Jongwon Lee, Jinho Park, and Minjoon Seo. How language models extrapolate outside the training data: A case study in textualized gridworld, 2024. URL https://arxiv.org/abs/2406.15275.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158.

OpenAI. Openai o1 system card. preprint, 2024.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought, 2023. URL https://arxiv.org/abs/2210.01240.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.
- Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning, 2022. URL https://arxiv.org/abs/2205.10816.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

# A FREQUENTLY ASKED QUESTIONS

- **Q1:** How does TSLM differ from Tree-of-Thought (ToT) inference? A1: While ToT applies external sampling and search algorithms to explore multiple reasoning paths at inference time, TSLM directly teaches the model to generate and explore tree-structured reasoning within a single generation process. This internal representation eliminates the need for multiple separate forward passes and external orchestration, leading to more efficient and effective exploration.
- **Q2:** Why doesn't standard supervised learning on linear Chain-of-Thought achieve similar extrapolation performance? A2: Standard CoT typically provides step-by-step verbal explanations without explicitly representing the global problem space or systematically exploring solution paths. TSLM differs by teaching models to construct and use complete tree-structured representations of the environment, enabling more robust planning in novel, complex scenarios.
- **Q3:** How does TSLM compare to reinforcement learning approaches like GRPO? A3: While GRPO and other RL approaches can achieve reasonable performance, particularly on open-ended tasks, they face challenges including complex reward engineering, computational intensity, and training instability. TSLM provides a simpler alternative that achieves comparable or better performance with straightforward supervised learning, particularly excelling on structured tasks like puzzle-solving.
- **Q4:** Does model size matter for TSLM? A4: Our experiments show that TSLM demonstrates consistent performance improvements across model sizes. Interestingly, TSLM helps smaller models achieve performance close to larger ones, suggesting that the structured tree representation provides a stronger learning signal that can compensate for model size limitations. However, larger models still benefit from TSLM, achieving the overall best performance.
- **Q5:** Can TSLM be applied to more complex real-world tasks? A5: While our current experiments focus on relatively constrained tasks, the principles of TSLM can extend to more complex domains. For tasks with well-defined structure (like code generation or game playing), direct application is straightforward. For more open-ended tasks, our bootstrapping approach provides a foundation for constructing synthetic tree-structured training data. Future work will explore applications to more diverse and complex reasoning domains.

# B RELATED WORK

**External Search-Augmented Language Models.** A major line of work augments language models with external search algorithms during inference. Tree-of-Thought Yao et al. (2023) applies breadth-first and depth-first search externally, sampling multiple reasoning paths with external evaluation. Graph-of-Thought Besta et al. (2024) extends this to general graph structures. Monte Carlo Tree Search approaches include RAP (Reasoning via Planning) Hao et al. (2023), which employs MCTS with world models, and LATS (Language Agent Tree Search) Zhou et al. (2023), which combines MCTS with reflection mechanisms. TS-LLM Feng et al. (2024) and AlphaCode-style approaches Li

 et al. (2022) integrate MCTS with language generation. These methods achieve strong performance but require multiple model invocations and external orchestration, limiting efficiency and integration with model training.

Multi-Path Generation and Reasoning. Various approaches explore multiple reasoning paths without structured search. Self-consistency decoding Wang et al. (2023) generates multiple independent reasoning paths and selects the most consistent answer. Ensemble methods combine predictions from multiple reasoning chains through diverse beam search and nucleus sampling variants. Recent models like of OpenAI (2024) and DeepSeek-R1 DeepSeek-AI et al. (2025) generate extended reasoning traces, but remain fundamentally sequential and may include redundant computation Chen et al. (2025). These approaches explore multiple paths but typically generate them independently without capturing structural relationships, unlike TSLM's coherent tree structures.

**Learning-Based Reasoning Enhancement.** Reinforcement learning has emerged as a dominant paradigm for improving reasoning capabilities. DeepSeek Math Shao et al. (2024) applies GRPO for mathematical reasoning, while other work uses RL from human feedback for instruction following. Actor-Critic methods and policy gradient approaches have shown promise in multi-step reasoning and mathematical problem solving. Algorithmic reasoning approaches train models to imitate procedures like sorting, graph traversal, and dynamic programming Kim et al. (2024); Yang et al. (2022), learning to execute classical algorithms step-by-step. However, these methods either require complex RL training or focus on single algorithmic traces rather than dynamic exploration strategies. TSLM demonstrates that carefully structured supervised learning can achieve comparable performance without RL's complexity.

#### C TSLM SERIALIZATION EXAMPLE

This section provides a detailed worked example showing how TSLM serializes tree structures for the Game of 24 task.

**Problem Setup.** Consider a Game of 24 problem with numbers [4, 5, 6, 10]. The goal is to find arithmetic operations that result in 24.

**Complete Tree Serialization.** The tree structure is serialized as:

```
[BOS] Step 1
4 + 5 = 9 [SEP1]
4 * 6 = 24 [SEP1] [SEP3]
Step 2
6 + 10 = 16 [SEP1]
6 * 10 = 60 [SEP1] [SEP3]
Step 3
9 + 16 = 25 [SEP2]
9 - 16 = -7 [SEP2] [SEP3] [EOS]
```

This format captures:

- Multiple candidate actions at each step
- Viability markers ([SEP1] for expandable, [SEP2] for dead ends)
- Step boundaries ([SEP3])
- Both successful and unsuccessful exploration paths

### D DETAILED LIMITATIONS AND FUTURE DIRECTIONS

This section provides an in-depth discussion of TSLM's current limitations and potential solutions.

**Computational Efficiency.** TSLM's node-wise training approach can be computationally inefficient compared to sequential methods. While the expanded supervision over tree nodes provides a richer training signal, the lack of computation caching means shared nodes are recomputed multiple times rather than reused. Future work could explore more efficient training strategies for scalability, including:

- Developing tree-aware attention mechanisms that can cache shared computations
- Exploring gradient accumulation strategies that account for tree structure
- Investigating sparse training approaches that selectively update tree components

**Supervision Quality.** The synthetic data generation framework's success heavily depends on the supervision model's capabilities and training distribution. For example, in GSM8K experiments, using Llama-3-8B as a supervision model produces poor-quality search trees. Different model families pose additional challenges - some generate Python code instead of reasoning steps, while others include extraneous dialogue that doesn't align with the desired reasoning format. We need to carefully select supervision models with output distributions matching the target reasoning scheme.

**Architectural Adaptation.** Current transformer architectures are not optimized for tree-structured generation. Developing architectures that naturally support tree-structured thinking could further enhance performance. Potential directions include hierarchical attention mechanisms and specialized positional encodings for tree structures.

**Application to Complex Domains.** Extending TSLM to more complex reasoning domains, including code generation, multi-hop reasoning, and creative problem-solving, represents an exciting direction for future work.

**Combining with RL.** While we've demonstrated TSLM's effectiveness with supervised learning alone, combining it with targeted reinforcement learning could potentially yield further improvements, particularly for adapting to user preferences or optimizing for specific outcomes.

# E DETAILED COMPARISON WITH ALGORITHMIC IMITATION

This section provides an in-depth comparison between TSLM and algorithmic imitation approaches.

**Representation Differences.** While algorithmic imitation approaches embed the complete reasoning process into a single sequential trace, TSLM explicitly represents multiple branching paths within a unified generation process. This fundamental difference enables:

- TSLM: Multiple potential paths represented simultaneously
- Algorithmic Imitation: Single linear trace representation

**Exploration Capabilities.** The exploration mechanisms differ significantly:

- TSLM: Enables systematic exploration of multiple branches with dynamic backtracking
- Algorithmic Imitation: Follows a predefined algorithmic path without exploration

**Adaptability and Flexibility.** TSLM provides greater adaptability:

- TSLM: Can dynamically adjust exploration strategy based on intermediate results
- Algorithmic Imitation: Executes a fixed procedure without adaptation

These distinctions enable TSLM to more effectively handle tasks requiring complex search or exploration, particularly when the optimal path is not immediately apparent.

# F PARAMETER EFFICIENCY DETAILED ANALYSIS

**Experimental Setup.** We conducted systematic experiments across the Qwen 2.5 model family (0.5B, 1.5B, 3B, 7B parameters) on Gridworld scaling tasks, training on  $10 \times 10$  grids and testing extrapolation to  $20 \times 20$  grids.

**Detailed Results.** Our experiments reveal striking parameter efficiency characteristics:

- TSLM scaling:  $0.5B (68\%) \rightarrow 1.5B (71\%) \rightarrow 3B (73\%) \rightarrow 7B (75\%)$
- Standard CoT:  $0.5B (15\%) \rightarrow 1.5B (18\%) \rightarrow 3B (21\%) \rightarrow 7B (26.5\%)$
- **GRPO**:  $0.5B (12.7\%) \rightarrow 1.5B (15\%) \rightarrow 3B (17\%) \rightarrow 7B (19\%)$
- **PC**:  $0.5B (3.0\%) \rightarrow 1.5B (3.5\%) \rightarrow 3B (5.5\%) \rightarrow 7B (6.0\%)$

**Analysis.** The flat scaling curve for TSLM ( $68\% \rightarrow 75\%$  from 0.5B to 7B) suggests that systematic exploration supervision captures *algorithmic competencies* largely independent of parameter count. In contrast, baseline methods show steeper parameter dependence, indicating reliance on brute-force memorization capacity. This fundamental difference demonstrates that TSLM teaches structured reasoning procedures that generalize across model scales.

# G DETAILED INFERENCE-TIME SCALING ANALYSIS

A key question is how TSLM performance scales with inference-time parameters like the number of candidate paths explored. We investigate different search strategies and their implications for model behavior. We compare three approaches for scaling inference:

- TSLM: Controls the number of candidates k by verifying the first k terminal states from Breadth-First Search (BFS) of the generated tree structure
- **Procedure cloning (PC)**: Verifies the first k terminal states from the sequential search trace
- Sequential cloning (SC): Uses Tree-of-Thought inference to generate the first k candidates via BFS.

### G.1 Performance Scaling with Number of Candidates

As shown in Figure 4, TSLM consistently outperforms baseline methods across different language model variants. With Llama-3-8B (solid lines), TSLM achieves 67.2% accuracy with just 5 candidates while PC and SC need 10 candidates to reach 63.6% and 61.9% respectively. Similar patterns emerge with Llama-3-8B-Instruct (dashed lines), where TSLM reaches 70.2% accuracy compared to 61.6% for PC and 67.9% for SC.

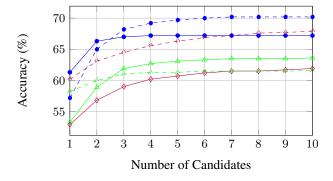


Figure 4: Performance scaling across different models and methods on GSM8K. Methods: TSLM (blue), PC (green), SC (purple). Models: Llama-3-8B (solid) and Llama-3-8B-Instruct (dashed).

### G.2 BFS vs DFS: Exploring Search Strategies

Within TSLM's inference framework, the choice between Breadth-First Search (BFS) and Depth-First Search (DFS) reveals fundamental differences in exploration priorities. We analyze these search strategies when TSLM's branching is guided by preference-based ordering.

**Theoretical Analysis.** Consider two distinct solution paths  $A = a_1, ..., a_n$  and  $B = b_1, ..., b_m$  in a preference-ordered search tree generated by TSLM.

BFS employs a dual prioritization strategy: it first considers path length (traversing A before B if n < m), then uses preference signals to break ties. When paths have equal length (n = m), BFS selects based on the first differing action's reward—if  $R(a_k) > R(b_k)$  at the earliest divergence point k, BFS favors path A. This means BFS prioritizes solution optimality first, using learned preferences to resolve ties.

In contrast, DFS operates purely on learned preference ordering, disregarding path length considerations. For the same paths A and B, DFS immediately follows the higher-reward action at any divergence point k, regardless of whether this leads to longer solution paths. This approach means DFS prioritizes learned preferences consistently, potentially sacrificing optimality for high-confidence actions.

**Empirical Results.** Figure 5 shows that DFS achieves better top-1 accuracy by finding preferred solutions first, but slightly underperforms BFS for top-3 to top-6 candidates since it does not prioritize optimality. However, both methods converge to the same accuracy as they traverse the complete tree. This validates our theoretical analysis that DFS provides faster convergence to high-confidence solutions while BFS offers better overall exploration coverage.

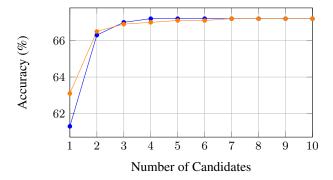


Figure 5: Comparison of BFS (blue) and DFS (orange) search strategies on Llama-3-8B.

### G.3 BASE MODEL PERFORMANCE IMPACT

We investigated whether the base model's performance influences TSLM's effectiveness by comparing performance across both Llama-3-8B and Llama-3-8B-Instruct variants. TSLM consistently outperforms baseline methods across both model variants, with improvements remaining robust across architectures. This indicates that TSLM's structured exploration provides consistent benefits independent of the base model.

### H TASK DETAILS AND EXAMPLES

#### H.1 GENERAL TASK SETUP DETAILS

Table 3 illustrates the general task settings for structured and open-ended tasks. Depth refers to the number of reasoning steps.

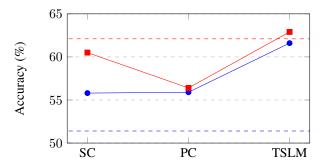


Figure 6: GSM8K accuracy across base models. Blue: Llama-3-8B, Red: Llama-3-8B-Instruct. Dashed lines show 4-shot performance.

Task	# Depth(Train)	# Depth(Test)	# Train Instance	# Test Instance	<b>Evaluation Metric</b>
Game of 24	3	3	1.2K	100	<b>Equation Validation</b>
Textualized Gridworld	$10 \times 10$	$10 \times 10$	10K	1.5K	Exact Matching
ProntoQA	1-5	1-5	4.5K	450	Exact Matching
GSM8K	2-9	2-11	7.5K	1.3K	Answer Matching

Table 3: General task settings

### H.2 TASK SCENARIO 1: STRUCTURED PLANNING

We first evaluate TSLM and baselines on two different structured planning tasks (Left in Figure 7). These tasks involve predefined search trees, enabling us to evaluate how well models reproduce algorithmic patterns:

**Game of 24** Given four numbers, the task is to obtain 24 with basic arithmetic operations  $(+, -, \times, \div)$ . For example, with inputs 8, 4, 3, and 6, a valid solution is  $(8+4) \times (6 \div 3) = 24$  Yao et al. (2023). Each action selects two operands and an operator, while states represent partial expressions. Since there may be multiple paths that reaches the answer (e.g.  $(8+4) \times (6 \div 3)$  and  $(6 \div 3) \times (4+8)$ ), we check if the final expression equals 24. We train each method on 1.2K instances and test on 100 instances.

**Textualized Gridworld** A text-based navigation task where an agent navigates a grid using cardinal directions (up, down, left, right) while avoiding obstacles. Given a start position (typically bottom left) and goal position (typically top right), the agent must find a valid path with shortest length. For instance, in a 3x2 grid with a pit at (1,1), the optimal solution is (right, right, up). Actions are single moves (up/down/left/right) and states are grid coordinates. We ensure all the environments to have

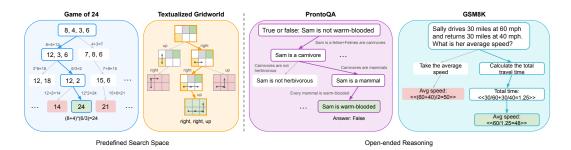


Figure 7: (Left) Structured planning tasks with predefined search spaces (e.g., Game of 24, Textualized Gridworld) where success is measured by the model's ability to reproduce algorithmic search patterns. (Right) General open-ended reasoning tasks (e.g., ProntoQA, GSM8K) requiring adaptive exploration of undefined search spaces.

unique shortest paths. We use 10K training instances with maximum grid size 10x10 and 1.5K test instances up to the same maximum size, evaluating exact path matching.

#### H.3 TASK SCENARIO 2: OPEN-ENDED REASONING TASKS

We next evaluate TSLM and baselines on two open-ended reasoning tasks (Right on Figure 7) requiring exploration of undefined solution spaces. We use the synthetic tree generation method from §4.2 for training.

**ProntoQA** ProntoQA Saparov & He (2023) is a logical reasoning dataset where models verify statement truth given premises. For example, given a premise "Every jompus is not small. Each impus is small. Each jompus is a dumpus. Alex is a jompus." and a query "True or false: Alex is not small?", the solution path to verify the query should be "Alex is a jompus. Every jompus is not small. Alex is not small. So the answer is True."

Actions are logical deduction steps and states are accumulated facts. With unique valid deduction paths, we verify whether the generated path is correct. We use 4.5K training and 450 test instances. For TSLM/PC, we use tree generation of beam search (k=5, temp=0.3) with Llama-3-8B as a base model for supervision.

**GSM8K** GSM8K Cobbe et al. (2021) is a mathematical word problem dataset requiring multi-step reasoning. For example, for a given problem "James runs 3 sprints 3 times a week. Each sprint is 60 meters. How many total meters per week?", one valid solution would be "3 sprints  $\times$  3 times = 9 sprints per week. 9 sprints  $\times$  60 meters = 540 meters. The answer is 540."

Actions are solution steps and states are partial solutions. Since multiple valid solution paths can reach the correct answer, we just validate whether the answer is correct. We use 7.5K training instances and 1.3K test instances. For TSLM/PC, we use tree generation of beam search (k=5, temp=0.1) with Llama-3-8B-Instruct as a base model for supervision.

### H.4 TASK EXAMPLES

For the following task examples, [SEP1] is a special token indicating that the action is good and can be further expanded, [SEP2] is a special token indicating that the action is not good and can't be expanded anymore, [SEP3] is a special token indicating the end of the current step, and [SEP4] is a special token indicating the success.

# **Gridworld SC Instance:**

**USER:** You are given a rectangular gridworld, where you can move up, down, left, or right as long as each of your x, y coordinate is within 0 to the x, y size of the grid. If you move up, your y coordinate increases by 1. If you move down, your y coordinate decreases by 1. If you move left, your x coordinate decreases by 1. If you move right, your x coordinate increases by 1.

You will interact with the girdworld environment to reach the goal state, while avoiding the pit and the wall. You cannot move through the wall or move outside the grid. If you fall into the pit, you lose. If you reach the goal, you win. For each of your turn, you will be given the possible moves.

You should respond your move with either one of 'up', 'down', 'left', or 'right'.

# ASSISTANT: OK

**USER:** Grid is from (7, 7) to (18, 13). Goal: (18, 13) Current: (7, 7) The pit is at (16, 9), (17, 7), (14, 8), (11, 9), and (16, 11). The wall is at (11, 7), (15, 10), (18, 12), (9, 9), (8, 7), (17, 10), (9, 12), (12, 7), (10, 13), (15, 12), (17, 9), (9, 11), (17, 12), (8, 9), (13, 11), (10, 9), (12, 9), (11, 11), (12, 12), (16, 10), (15, 8), (9, 7), (8, 11), (7, 13), (13, 13), and (13, 10).

# ASSISTANT:

```
1022 Step 1
1023 up
1024 (7, 8)[SEP1][SEP3]
1025 Step 2
```

up

```
1026
             (7, 9)[SEP1][SEP3]
1027
             Step 3
1028
             up
1029
             (7, 10)[SEP1][SEP3]
1030
             Step 4
1031
             (7, 11)[SEP1][SEP3]
1032
1033
```

1035

1036

1037

1038

1039

1040

1041

1043

1044

1045

1046

1047

1067

1068

1069

1070

1071

1072

1074

1075

1077

1078

1079

### **Gridworld PC Instance:**

**USER:** You are given a rectangular gridworld, where you can move up, down, left, or right as long as each of your x, y coordinate is within 0 to the x, y size of the grid. If you move up, your y coordinate increases by 1. If you move down, your y coordinate decreases by 1. If you move left, your x coordinate decreases by 1. If you move right, your x coordinate increases by 1.

You will interact with the girdworld environment to reach the goal state, while avoiding the pit and the wall. You cannot move through the wall or move outside the grid. If you fall into the pit, you lose. If you reach the goal, you win. For each of your turn, you will be given the possible moves.

You should respond your move with either one of 'up', 'down', 'left', or 'right'.

### **ASSISTANT: OK**

**USER:** Grid is from (7, 7) to (18, 13). Goal: (18, 13) Current: (7, 7) The pit is at (16, 9), (17, 7), (14, 8), (11, 9), and (16, 11). The wall is at (11, 7), (15, 10), (18, 12), (9, 9), (8, 7), (17, 10), (9, 12), (12, 7), (10, 13), (15, 12), (17, 9), (9, 11), (17, 12), (8, 9), (13, 11), (10, 9),(12, 9), (11, 11), (12, 12), (16, 10), (15, 8), (9, 7), (8, 11), (7, 13), (13, 13), and (13, 10).

#### **ASSISTANT:**

```
1048
             Step 1
1049
             up
1050
             (7, 8)[SEP1]
1051
             down
1052
             (7, 6)[SEP2]
1053
             left
1054
             (6,7)[SEP2]
1055
             right
1056
             (8, 7)[SEP2][SEP3]
1057
             Step 2
1058
             (7, 9)[SEP1]
1059
             down
             (7, 7)[SEP2]
1061
             left
1062
             (6, 8)[SEP2]
1063
             right
1064
             (8, 8)[SEP1][SEP3]
```

## **Gridworld TSLM Instance:**

**USER:** You are given a rectangular gridworld, where you can move up, down, left, or right as long as each of your x, y coordinate is within 0 to the x, y size of the grid. If you move up, your y coordinate increases by 1. If you move down, your y coordinate decreases by 1. If you move left, your x coordinate decreases by 1. If you move right, your x coordinate increases by 1.

You will interact with the girdworld environment to reach the goal state, while avoiding the pit and the wall. You cannot move through the wall or move outside the grid. If you fall into the pit, you lose. If you reach the goal, you win. For each of your turn, you will be given the possible moves.

You should respond your move with either one of 'up', 'down', 'left', or 'right'.

# ASSISTANT: OK

**USER:** Grid is from (7, 7) to (18, 13). Goal: (18, 13) Current: (7, 7) The pit is at (16, 9), (17, 7), (14, 8), (11, 9), and (16, 11). The wall is at (11, 7), (15, 10), (18, 12), (9, 9), (8, 7), (17, 10), (9, 12), (12, 7), (10, 13), (15, 12), (17, 9), (9, 11), (17, 12), (8, 9), (13, 11), (10, 9),(12, 9), (11, 11), (12, 12), (16, 10), (15, 8), (9, 7), (8, 11), (7, 13), (13, 13), and (13, 10).

```
1080
            ASSISTANT:
1081
            Step 1
1082
            up
1083
            (7, 8)[SEP1][SEP3]
1084
            Step 2
1085
            up
            (7, 9)[SEP1][SEP3]
1086
            Step 3
1087
            up
1088
            (7, 10)[SEP1][SEP3]
1089
            Step 4
1090
            up
1091
            (7, 11)[SEP1]
1092
            down
1093
            (7, 9)[SEP2]
1094
            left
1095
            (6, 10)[SEP2]
            right
1096
            (8, 10)[SEP1][SEP3]
1097
1098
            Game24 SC Instance:
1099
            USER: Input: 4, 5, 6, 10
1100
            ASSISTANT: Step 1
            4 + 5 = 9[SEP1]
1101
            Step 2
1102
            6 + 10 = 16[SEP1][SEP3]
1103
1104
            Game24 PC Instance:
            USER: Input: 4, 5, 6, 10
1105
            ASSISTANT: Step 1
1106
            4 + 5 = 9[SEP1]
1107
            4 - 5 = -1[SEP1]
1108
            4 * 5 = 20[SEP1]
1109
            4/5 = 4/5[SEP1]
1110
            4 + 6 = 10[SEP1]
1111
            4 - 6 = -2[SEP1]
1112
            4 * 6 = 24[SEP1]
1113
            4/6 = 2/3[SEP1]
1114
            4 + 10 = 14[SEP1]
1115
            4 - 10 = -6[SEP1]
1116
            4 * 10 = 40[SEP1]
            4/10 = 2/5[SEP1]
1117
            5 + 4 = 9[SEP1]
1118
            5 - 4 = 1[SEP1]
1119
            5 * 4 = 20[SEP1]
1120
            5/4 = 5/4[SEP1]
1121
            5 + 6 = 11[SEP1]
1122
            5 - 6 = -1[SEP1]
1123
            5 * 6 = 30[SEP1]
1124
            5/6 = 5/6[SEP1]
1125
            5 + 10 = 15[SEP1]
1126
            5 - 10 = -5[SEP1]
1127
            5 * 10 = 50[SEP1]
            5/10 = 1/2[SEP1]
1128
            6 + 4 = 10[SEP1]
1129
            6 - 4 = 2[SEP1]
1130
            6 * 4 = 24[SEP1]
1131
            6/4 = 3/2[SEP1]
1132
            6 + 5 = 11[SEP1]
1133
            6 - 5 = 1[SEP1]
```

```
1134
            6 * 5 = 30[SEP1]
1135
            6/5 = 6/5[SEP1]
1136
            6 + 10 = 16[SEP1]
            6 - 10 = -4[SEP1]
1137
1138
            6 * 10 = 60[SEP1]
            6/10 = 3/5[SEP1]
1139
             10 + 4 = 14[SEP1]
1140
             10 - 4 = 6[SEP1]
1141
             10 * 4 = 40[SEP1]
1142
             10/4 = 5/2[SEP1]
1143
             10 + 5 = 15[SEP1]
1144
             10 - 5 = 5[SEP1]
1145
             10 * 5 = 50[SEP1]
1146
             10/5 = 2[SEP1]
1147
             10 + 6 = 16[SEP1]
1148
             10 - 6 = 4[SEP1]
1149
             10 * 6 = 60[SEP1]
             10 / 6 = 5/3[SEP1][SEP3]
1150
            Step 2
1151
            6 + 10 = 16[SEP1]
1152
            6 - 10 = -4[SEP1]
1153
            6 * 10 = 60[SEP1]
1154
            6/10 = 3/5[SEP1]
1155
            6 + 9 = 15[SEP1]
1156
            6 - 9 = -3[SEP1]
1157
            6*9 = 54[SEP1]
1158
            6/9 = 2/3[SEP1]
1159
             10 + 6 = 16[SEP1]
1160
             10 - 6 = 4[SEP1]
             10 * 6 = 60[SEP1]
1161
             10/6 = 5/3[SEP1]
1162
             10 + 9 = 19[SEP1]
1163
             10 - 9 = 1[SEP1]
1164
             10 * 9 = 90[SEP1]
1165
             10/9 = 10/9[SEP1]
1166
            9 + 6 = 15[SEP1]
1167
             9 - 6 = 3[SEP1]
1168
            9 * 6 = 54[SEP1]
1169
            9/6 = 3/2[SEP1]
1170
             9 + 10 = 19[SEP1]
            9 - 10 = -1[SEP1]
1171
            9 * 10 = 90[SEP1]
1172
            9/10 = 9/10[SEP1][SEP3]
1173
            6 + 10 = 16[SEP1]
1174
            6 - 10 = -4[SEP1]
1175
            6 * 10 = 60[SEP1]
1176
            6/10 = 3/5[SEP1]
1177
            6 + -1 = 5[SEP1]
1178
            6 - -1 = 7[SEP1]
1179
            6 * -1 = -6[SEP1]
1180
            6/-1 = -6[SEP1]
1181
             10 + 6 = 16[SEP1]
             10 - 6 = 4[SEP1]
1182
             10 * 6 = 60[SEP1]
1183
             10/6 = 5/3[SEP1]
1184
             10 + -1 = 9[SEP1]
1185
             10 - -1 = 11[SEP1]
1186
             10 * -1 = -10[SEP1]
1187
             10 / -1 = -10[SEP1]
```

```
1188
            -1 + 6 = 5[SEP1]
1189
            -1 - 6 = -7[SEP1]
1190
            -1 * 6 = -6[SEP1]
            -1/6 = -1/6[SEP1]
1191
1192
            -1 + 10 = 9[SEP1]
            -1 - 10 = -11[SEP1]
1193
            -1 * 10 = -10[SEP1]
1194
            -1/10 = -1/10[SEP1][SEP3]
1195
            6 + 10 = 16[SEP1]
1196
            6 - 10 = -4[SEP1]
1197
            6 * 10 = 60[SEP1]
1198
            6/10 = 3/5[SEP1]
1199
            6 + 20 = 26[SEP1]
1200
            6 - 20 = -14[SEP1]
1201
            6 * 20 = 120[SEP1]
1202
            6/20 = 3/10[SEP1]
             10 + 6 = 16[SEP1]
1203
             10 - 6 = 4[SEP1]
1204
             10 * 6 = 60[SEP1]
1205
             10/6 = 5/3[SEP1]
1206
             10 + 20 = 30[SEP1]
1207
             10 - 20 = -10[SEP1]
1208
             10 * 20 = 200[SEP1]
1209
             10/20 = 1/2[SEP1]
1210
            20 + 6 = 26[SEP1]
1211
            20 - 6 = 14[SEP1]
1212
            20 * 6 = 120[SEP1]
1213
            20 / 6 = 10/3[SEP1]
1214
             20 + 10 = 30[SEP1]
            20 - 10 = 10[SEP1]
1215
            20 * 10 = 200[SEP1]
1216
            20 / 10 = 2[SEP1][SEP3]
1217
1218
            Game24 TSLM Instance:
1219
            USER: Input: 4, 5, 6, 10
1220
            ASSISTANT: Step 1
            4 + 5 = 9[SEP1]
1221
            Step 2
1222
            6 + 10 = 16[SEP1]
1223
            6 - 10 = -4[SEP1]
1224
            6 * 10 = 60[SEP1]
1225
            6/10 = 3/5[SEP1]
1226
            6 + 9 = 15[SEP1]
1227
            6 - 9 = -3[SEP1]
1228
            6 * 9 = 54[SEP1]
1229
            6/9 = 2/3[SEP1]
1230
             10 + 6 = 16[SEP1]
             10 - 6 = 4[SEP1]
1231
             10 * 6 = 60[SEP1]
1232
             10/6 = 5/3[SEP1]
1233
             10 + 9 = 19[SEP1]
1234
             10 - 9 = 1[SEP1]
1235
             10 * 9 = 90[SEP1]
1236
             10/9 = 10/9[SEP1]
1237
            9 + 6 = 15[SEP1]
1238
            9 - 6 = 3[SEP1]
1239
            9*6 = 54[SEP1]
1240
            9/6 = 3/2[SEP1]
            9 + 10 = 19[SEP1]
```

```
1242
             9 - 10 = -1[SEP1]
1243
             9*10 = 90[SEP1]
             9/10 = 9/10[SEP1][SEP3]
1245
             ProntoQA SC Instance:
1246
             USER: Arthropods are small. Insects are not eight-legged. Lepidopterans are insects.
1247
             Invertebrates are animals. Every spider is eight-legged. Insects are arthropods. Every
1248
            butterfly is a lepidopteran. Animals are multicellular. Arthropods are invertebrates. Sam is
1249
            a butterfly.
1250
            True or false: Sam is not eight-legged.
1251
             ASSISTANT:
1252
             Step 1
             Sam is a butterfly.[SEP1][SEP3]
1253
             Step 2
1254
            Every butterfly is a lepidopteran.[SEP1][SEP3]
1255
             Step 3
1256
             Sam is a lepidopteran.[SEP1][SEP3]
1257
             Step 4
1258
             Lepidopterans are insects.[SEP1][SEP3]
1259
             Step 5
1260
             Sam is an insect.[SEP1][SEP3]
1261
             Step 6
1262
            Insects are not eight-legged.[SEP1][SEP3]
             Step 7
1263
             Sam is not eight-legged.[SEP1][SEP3]
1264
             Step 8
1265
             The answer is true.[SEP4][SEP3]
1266
             ProntoQA PC Instance: c Arthropods are small. Insects are not eight-legged. Lepi-
1267
             dopterans are insects. Invertebrates are animals. Every spider is eight-legged. Insects are
1268
             arthropods. Every butterfly is a lepidopteran. Animals are multicellular. Arthropods are
1269
             invertebrates. Sam is a butterfly.
1270
             True or false: Sam is not eight-legged.
1271
             ASSISTANT:
1272
             Step 1
1273
             Sam is a butterfly.[SEP1][SEP3]
1274
             Step 2
1275
            Every butterfly is a lepidopteran.[SEP1][SEP3]
1276
             Step 3
1277
             Sam is a lepidopteran.[SEP1][SEP3]
1278
             Step 4
             Lepidopterans are insects.[SEP1][SEP3]
1279
             Step 5
1280
             Sam is an insect.[SEP1][SEP3]
1281
1282
             Insects are not eight-legged.[SEP1]
1283
             Insects are arthropods.[SEP1][SEP3]
1284
             Step 7
1285
             Sam is not eight-legged.[SEP1][SEP3]
1286
             Sam is an arthropod.[SEP1][SEP3]
1287
            ProntoQA TSLM Instance:
1288
             USER: Arthropods are small. Insects are not eight-legged. Lepidopterans are insects.
1289
             Invertebrates are animals. Every spider is eight-legged. Insects are arthropods. Every
1290
             butterfly is a lepidopteran. Animals are multicellular. Arthropods are invertebrates. Sam is
1291
             a butterfly.
            True or false: Sam is not eight-legged.
1293
             ASSISTANT:
1294
             Step 1
             Sam is a butterfly.[SEP1][SEP3]
1295
```

```
1296
             Step 2
1297
            Every butterfly is a lepidopteran.[SEP1][SEP3]
1298
             Step 3
1299
             Sam is a lepidopteran.[SEP1][SEP3]
1300
            Step 4
            Lepidopterans are insects.[SEP1][SEP3]
1301
            Step 5
1302
             Sam is an insect.[SEP1][SEP3]
1303
             Step 6
1304
            Insects are not eight-legged.[SEP1][SEP3]
1305
             Step 7
1306
            Sam is not eight-legged.[SEP1]
1307
            Sam is an arthropod.[SEP1][SEP3]
             GSM8K SC Instance:
1309
             USER: Kylar went to the store to buy glasses for his new apartment. One glass costs $5,
1310
            but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How
1311
            much does he need to pay for them?
1312
            ASSISTANT:
1313
            Step 1
1314
            Every second glass costs 5*0.6 = \$ << 5*0.6 = 3 >> 3.[SEP1][SEP3]
1315
             Step 2
             So for 16 glasses, Kylar needs to pay 3*8 = $ << 16/2 = 8 >> for every second
1316
             glass.[SEP1][SEP3]
1317
1318
             GSM8K PC Instance:
1319
             USER: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts
1320
            in total does it take?
1321
            ASSISTANT:
            Step 1
1322
            It takes 2 * 0.5 = << 2 * 0.5 = 1 >> 1 bolt of white fiber.[SEP1]
1323
            The robe takes 2 bolts of blue fiber.[SEP1][SEP3]
1324
            Step 2
1325
            So it takes 2 + 1 = << 2 + 1 = 3 >> 3 bolts in total.[SEP1][SEP3]
1326
            It also takes half as much white fiber, which means it takes 1 bolt of white fiber (since half
1327
            of 2 is 1).[SEP1][SEP3]
1328
             Step 3
            The answer is 3.[SEP4][SEP3]
             GSM8K TSLM Instance:
1331
             USER: Kylar went to the store to buy glasses for his new apartment. One glass costs $5,
1332
            but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How
1333
            much does he need to pay for them?
1334
            ASSISTANT:
1335
            Step 1
1336
            Every second glass costs 5 * 0.6 = \$ << 5 * 0.6 = 3 >> 3.[SEP1][SEP3]
1337
            So for 16 glasses, Kylar needs to pay 3*8 = $ << 16/2 = 8 >>  for every second
1338
            glass.[SEP1]
1339
            So for 8 glasses, it would cost 8 \times 3 = 24 dollars. For the remaining 8 glasses, it would
1340
            \cos 8 \times 5 = 40 dollars. So in total, it would \cos 24 + 40 = 64 dollars. [SEP1][SEP3]
1341
1342
```

# I USE OF LARGE LANGUAGE MODELS

1343

1344 1345

1347 1348

1349

We acknowledge the use of large language models to assist in the preparation of this manuscript. Specifically:

**Writing assistance.** Large language models were used to aid in polishing and refining the writing throughout the paper, including improving clarity, grammar, and expression of technical concepts.

**Related work discovery.** Large language models were employed for retrieval and discovery tasks, particularly in identifying and organizing relevant related work and ensuring comprehensive coverage of the literature.

All technical contributions, experimental design, implementation, analysis, and conclusions presented in this work are the original work of the authors. The use of LLMs was limited to editorial assistance and literature search support, and did not influence the core scientific contributions or findings reported in this paper.