Extended Abstract

Motivation

Method

Implementation

Results

Discussion

Conclusion

In-context Search: Efficiency Boost or Fundamentally New Capability?

Angel Raychev
Department of Computer Science
Stanford University
angelray@stanford.edu

Yalcin Tur
Department of Computer Science
Stanford University
yalcintr@stanford.edu

Mihajlo Stojkovic
Department of Computer Science
Stanford University
mstojkov@stanford.edu

1 Introduction

Reasoning models, most notably OpenAI's O1 OpenAI (2024), represent a novel category of large language models (LLMs) characterized by the incorporation of in-context search during generation. This approach uniquely enables scaling inference-time computation to directly improve model performance. While classical search methods also leverage additional computational resources to enhance standard LLM performance, reasoning models employing in-context search typically demonstrate superior computational efficiency, achieving comparable or improved results with significantly reduced computational cost.

Traditional language models estimate the probability of producing an answer a given a question q as follows:

$$\pi_{\text{data}}(a \mid q) \propto \int \pi_{\text{data}}(a \mid q s_1 \dots s_n) \prod_{i=1}^{n} \pi_{\text{data}}(s_i \mid q s_1 \dots s_{i-1})\, dS \tag{1}$$

where $s_1, s_2, \dots, s_n$ represent intermediate solution steps. In contrast, reasoning models using in-context search formulate generation differently Xiang et al. (2025):

$$\pi_{\text{data}}(s_1 \dots s_n a \mid q) \propto \int \pi_{\text{data}}(s_1 \dots s_n a \mid q z_1 \dots z_K) \prod_{i=1}^{K} \pi_{\text{data}}(z_i \mid q z_1 \dots z_{i-1})\, dZ \tag{2}$$

Here, $z_1, z_2, \dots, z_K$ represent intermediate, model-generated reasoning steps leading to the final solution $s_1, \dots, s_n$. Structured search methods impose explicit structure on these intermediate steps. In-context search models, however, do not enforce internal structure during training but instead guide these intermediate reasoning steps externally.

A significant open question raised by Xiang et al. (2025) is whether reasoning models utilizing in-context search merely provide greater computational efficiency compared to structured search methods or if they indeed possess the capability to solve fundamentally new classes of problems that classical structured search inherently cannot, regardless of available compute. To answer this, analyzing scaling laws for these methods becomes essential.

2 Related Work

Classical search. Structured-reasoning with LLMs has been tackled via prompt-based tree search methods such as Tree-of-Thought Yao et al. (2023) and Reasoning via Planning Hao et al. (2023),

which use a frozen language model as a value function and perform BFS or DFS. These methods rely on prompt engineering and self-verification by the generator, leading to brittleness, and their shallow search constrains them to simple tasks. By contrast, AlphaZero–style Monte Carlo Tree Search trains a dedicated policy network alongside a separate value head, enabling deep, scalable exploration without depending on the model's own probabilities. Empirically, this critic-based MCTS outperforms prompt-based verifiers and shallow tree search. MCTS seems the best of structured search, and that's why we use it following the TS-LLM methodology Feng et al. (2023).

In-context search. Parallel to explicit search, a new family of *in-context* models has emerged (OpenAI O1 OpenAI (2024), DeepSeek R1 DeepSeek-AI (2025), xAI Grok-3 Reuters (2025), etc.). Unlike classical approaches, these systems solve problems by iterating entire trajectories inside the prompt window. The first systematic recipe is Stream of Search (SoS) Gandhi et al. (2024), which trains on the *full* search traversals, not merely the final answers, and then refines the model via Self-Taught Reasoner (STaR) Zelikman et al. (2022). Our work follows SoS for the in-context branch while adopting an AlphaZero-inspired MCTS for the classical branch, enabling a controlled head-to-head comparison.

3 Methods

Classical search (MCTS). We adopt Monte Carlo Tree Search (MCTS), similar to AlphaZero Feng et al. (2023). Instead of using roll-outs, we train a policy π_θ and a value function v_ϕ in together. For every question q, MCTS produces a trajectories $s_1 s_2 \dots s_n a$ that ends with a parsable answer a. Positive trajectories (those whose answer is correct) fine-tune the policy, while all terminal trajectories supervise the value network in a binary-classification objective:

$$\mathcal{L}_{\text{policy}} = \mathbb{E}_{q,s,a\sim\mathcal{D}^+}\big[-\log \pi_\theta(s_1 \dots s_n a \mid q)\big], \tag{3}$$

$$\mathcal{L}_{\text{value}} = \mathbb{E}_{q,s,a\sim\mathcal{D}}\Big[\ell_{\text{BCE}}\big[v_\phi(s_1 \dots s_n a \mid q),\, r(a \mid q)\big] + \sum_{t=1}^{n} \ell_{\text{BCE}}\big[v_\phi(s_1 \dots s_t \mid q),\, r(a \mid q)\big]\Big], \tag{4}$$

where $\mathcal{D}$ is the distribution of all terminal traces and $\mathcal{D}^+$ the subset of positives. The binary reward $r(a \mid q) \in \{0, 1\}$ is provided by an oracle grader. Training alternates between (i) data collection with the current MCTS policy/value and (ii) parameter updates via (3)–(4).

MCTS details. Each search cycle has four phases—selection, expansion, evaluation, back-propagation.

Selection. At depth i we pick the next action (sentence-level)

$$s_i^\star = \arg\max_b U(s_1 \dots s_{i-1} s_i^{(b)}), \qquad U = Q + c_{\text{explore}} \frac{\sqrt{N(s_1 \dots s_{i-1})}}{N(s_1 \dots s_{i-1} s_i^{(b)}) + 1},$$

where $Q(\cdot)$ is the running average return and $N(\cdot)$ the visit count. We continue until reaching a leaf or forming a complete answer $s_1 \dots s_n a$.

Expansion & evaluation. From a leaf we sample B children

$$s_i^{(1)}, \dots, s_i^{(B)} \sim \pi_\theta(\cdot \mid q s_1 \dots s_{i-1}),$$

evaluate each with v_ϕ, initialise $N = 0$ and $Q = v_\phi$, and skip this step if the node is already terminal.

Back-propagation. Traversing back to the root, we update visit counts and running means:

$$Q(s_1 \dots s_{j-1}) \leftarrow Q(s_1 \dots s_{j-1}) + \frac{v_\phi(s_1 \dots s_{j-1} \mid q) - Q(s_1 \dots s_{j-1})}{N(s_1 \dots s_{j-1}) + 1}, \quad j \le i.$$

Search stops when the compute budget is exhausted. During training we store only answer-yielding paths; at inference we output the most-visited branch based on the Q-values.

In-context search (SoS + STaR). After convergence we generate full tree traversals with the final π_θ, v_ϕ, flatten them, and train a fresh in-context model $\overline{\pi}_\psi$ from scratch, following Stream of Search Gandhi et al. (2024). Traversals of multiple tree sizes serve as supervised data, and the model is also trained to emit the final answer. We then apply STaR for an additional performance boost.

4 Implementation Details

The complete source code is available at `https://github.com/RaychevAngel/classical_vs_contextual_search`. All components were implemented from scratch. Although an open-source MCTS existed, its lack of clean parallelism motivated a fresh implementation. Similarly, the original Stream-of-Search (SoS) codebase was small but not fully aligned with our workflow, so we re-implemented it to keep the repository self-contained.

Model architecture. We employ Qwen-2.5 checkpoints throughout. The policy π_θ and value v_ϕ share the 1.5 B parameter base, whereas the in-context model $\overline{\pi}_\psi$ starts from the 3 B variant. Both approaches therefore use an equal 3 B total parameters drawn from the same family, ensuring a fair comparison.

Task and dataset. Our benchmark generalises *Game 24*: given four integers $x_1, \ldots, x_4 \in [1, 12]$ and a target $t \in [1, 999]$, build t exactly once using $\{+, -, \times, \div\}$ and parentheses. We generated 105 000 problems and split them into 99 000/3 000/3 000 for train/val/test. To prime the policy, we hand-crafted solutions for the first 9 000 training instances, instructing the model to emit each step on a new line and to format the final answer parse-ably.

```
q: Use 1, 4, 8, 11 to make 308.
s1: 8-1=7 Left: 4, 7, 11
s2: 4*7=28 Left: 11, 28
s3: 11*28=308 Left: 308
a: The  answer is: 4*(8-1)*11=308.
```

```
q: Use 1, 3, 8, 12 to make 32.
s1: 3-1=2 Left: 2, 8, 12
s2: 2*12=24 Left: 8, 24
s3: 8+24=32 Left: 32
a: The answer is: (3-1)*12+8=32.
```

```
q: Use 5, 7, 10, 11 to make 339.
s1: 5*7=35 Left: 10, 35, 11
s2: 10*35=350 Left: 11, 350
s3: 350-11-339 Left: 339
a: The answer is: 5*7*10-11=339.
```

Bootstrapping. The base policy π_θ^0 is fine-tuned for one epoch on the synthetic solutions (LR = 5×10^{-5}, batch = 256) to obtain π_θ^1. Pre-training v_ϕ on synthetic negatives is possible but brittle—enumerating the many ways a derivation can fail often traps the network in local minima—so we skip this step.

Iterative MCTS training. At iteration $i \geq 1$ we: First, we sample 9 000 unseen training questions. Next, we grow one tree per question using a branch factor of 5, up to 25 expansions, and an exploration constant $c_{\text{explore}} = 0.3$. Finally, we collect all terminal traces and use them as the next minibatch. vLLM serves both models; the policy uses temperature 1.0 and top-5 nucleus sampling, filtering duplicates. For the value head we take a designated logit and apply a sigmoid.

Large trees raise diversity and help learning; with more time we would explore branch factors 10–20. Each round trains π_θ^i (batch = 256) and v_ϕ^i (batch = 1024) for one epoch, starting at LR $= 2 \times 10^{-5}$ and decaying to 5×10^{-6} by iteration 6, where performance plateaus. Intermediate-state labels are *soft*: instead of the binary reward $r(a \mid q)$ we use the average final reward reachable from that node, reducing gradient noise without changing optima (cf. Eq. 4).

In-context model (SoS). We freeze π_θ^6, v_ϕ^6, sample 12 000 fresh questions, and record complete MCTS traversals. States are logged in first-visit order, then flattened into a single string; the Q-optimal solution and its answer (if correct) are appended. Extra markers help the model navigate the tree.

```
q: Use 3, 3, 5, 12 to make 51.
<START_THOUGHT>
z1: N1->Q | 3*12=36 Left: 3, 5, 36
z2: N2->Q | 12*5=60 Left: 3, 3, 60
z3: N3->Q | 5*12=60 Left: 3, 3, 60
z4: N4->N1 | 3*5=15 Left: 15, 36
z5: N5->N2 | 60-3=57 Left: 3, 57
z6: N6->N3 | 3+3=6 Left: 6, 60
z7: N7->N1 | 5*3=15 Left: 15, 36
z8: N8->N3 | 60-3=57 Left: 3, 57
z9: N9->N2 | 3*3=9 Left: 9, 60
z10: N10->N4 | 36+15=51 Left: 51
z11: N11->N10->N4->N1->Q | 3*12+3*5= 51
z12: N12->N7 | 36+15=51 Left: 51
z13: N13->N12->N7->N1->Q | 3*12+5*3= 51
z14: N14->N3 | 3*3=9 Left: 9, 60
z15: N15->N5 | 57-3=54 Left: 54
z16: N16->N6 | 60-6=54 Left: 54
z17: N17->N9 | 60-9=51 Left: 51
z18: N18->N17->N9->N2->Q | 12*5-3*3= 51
z19: N19->N8 | 57-3=54 Left: 54
z20: N20->N15->N5->N2->Q | 12*5-3-3= 54
z21: N21->N14 | 60-9=51 Left: 51
z22: N22->N21->N14->N3->Q | 5*12-3*3= 51
z23: N23->N19->N8->N3->Q | 5*12-3-3= 54
<END_THOUGHT>
<START_ANSWER>
a: 3*12+3*5=51
<END_ANSWER>
```

Traversals are generated with branch factors $\{1, 2, 3, 5, 8, 13, 21\}$. A separate $\overline{\pi}_\psi$ is trained for each factor (16 epochs, LR $= 2 \times 10^{-5}$, batch = 64).

STaR refinement. We apply two STaR variants to the best $\overline{\pi}_\psi$: We compare two STaR variants on 12 000 new questions. In the first (STaR), we generate exactly one trajectory per question. In

the second (STaR), we generate up to ten trajectories for each question and select the first one that produces a correct answer. Both converge in a single iteration, producing $\overline{\pi}_{\psi}^{\mathrm{STaR}}$ and $\overline{\pi}_{\psi}^{\mathrm{STaR}^*}$.

5 Experiments and Results

We retain five MCTS checkpoints (the initial policy-only run is omitted, as it lacks a value head) and 3×7 SoS checkpoints—$\overline{\pi}_{\psi}$, $\overline{\pi}_{\psi}^{\mathrm{STaR}}$, and $\overline{\pi}_{\psi}^{\mathrm{STaR}^*}$—for every branch factor $b \in \{1, 2, 3, 5, 8, 13, 21\}$. All models are evaluated on the held-out test set; results appear in Figures 1 and 2.

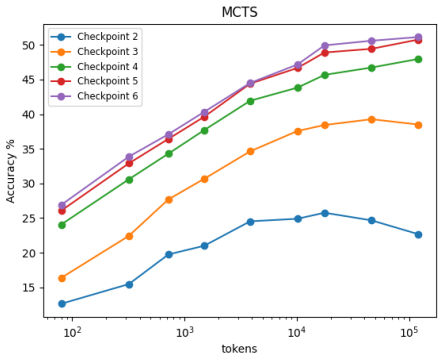

Figure 1: MCTS at different checkpoints

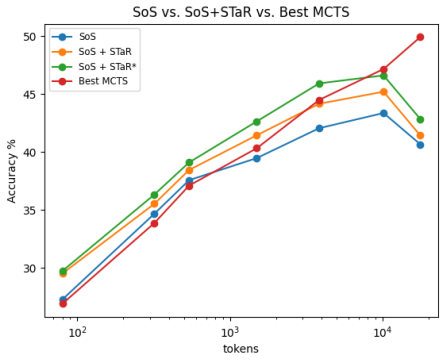

Figure 2: Classical vs. In-Context Search

MCTS get's up to 51.13% accuracy at its final checkpoint, improving steadily as the compute budget grows.The in-context search variants outperform MCTS at small and medium budgets, yet their accuracy drops abruptly beyond $\sim$ 20,000 tokens—likely a context-length limitation—so MCTS ultimately scales better.

6 Discussion and Conclusion

Our study set out to clarify whether in-context search confers *intrinsically* greater reasoning power or merely a more efficient use of compute relative to classical structured search. The evidence so far is nuanced.

Compute–accuracy trade-off. Stream-of-Search (SoS) beats policy–value MCTS at very small inference budgets, but the advantage vanishes once the budget rises: SoS plateaus, whereas MCTS keeps climbing and ultimately overtakes. Hence, the apparent efficiency of in-context search does not translate into better asymptotic performance under our current training regime.

Limitations of our SoS training. We applied only a single STaR pass—essentially a supervised warm-start. The model was never *reinforced* to avoid poor trajectories. Incorporating policy-gradient updates (e.g. PPO, DPO, GRPO) or contrastive penalties against dead-ends could push the in-context model much further up the compute curve.

Task bias. Generalised GAME 24 favours classical search:

- a sparse, well-typed action space simplifies semantic filtering;
- trees are shallow, so MCTS explores them exhaustively;
- vector search over known formulas already yields strong priors.

Moving to broader mathematical domains—with larger vocabularies, longer derivations, and looser syntax—should accentuate the flexibility of in-context reasoning.

Context-length ceiling. The sharp drop in SoS accuracy beyond $\sim$20 k prompt tokens almost certainly stems from context-length limits. Techniques such as retrieval-augmented prompting, segment-wise compression, or hierarchical deliberation may alleviate this barrier.

Outlook. While MCTS remains the most scalable *structured* search in our experiments—supporting the TS-LLM philosophy—the question of whether in-context search can ultimately outstrip external search is still open. Future work should:

1. integrate stronger RL objectives into SoS training,
2. evaluate on tasks with deeper reasoning chains,
3. test longer-context or memory-augmented language models.

Only with such advances can we definitively determine whether in-context search unlocks qualitatively new capabilities or simply reshuffles the compute–performance frontier.

7 Team Contributions

- **Angel Raychev:** Lead developer. Implemented most of the codebase, repeatedly refactored the MCTS engine, and ran all experiments.
- **Yalcin Tur:** Contributed to the initial code base.
- **Mihajlo Stojković:** Contributed to the initial code base.

Note. This work extends our CS224N project. All three authors built the first MCTS prototype, after which Yalcin and Mihajlo stepped away. Angel has since rewritten the entire codebase several times, fixing numerous bugs and spending hundreds of hours tuning MCTS hyper-parameters. The Stream-of-Search module is completely new and introduced further engineering challenges. Development is still ongoing.

References

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025). https://doi.org/10.48550/arXiv.2501.12948

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179* (2023).

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. 2024. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683* (2024).

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 8154–8173. https://doi.org/10.18653/v1/2023.emnlp-main.507

OpenAI. 2024. OpenAI o1 System Card. *arXiv preprint arXiv:2412.16720* (2024). https://doi.org/10.48550/arXiv.2412.16720

Reuters. 2025. Musk's xAI unveils Grok-3 AI chatbot to rival ChatGPT, China's DeepSeek. https://www.reuters.com/technology/artificial-intelligence/musks-xai-unveils-grok-3-ai-chatbot-rival-chatgpt-chinas-deepseek-2025-02-18/.

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, et al. 2025. Towards System 2 Reasoning in LLMs: Learning How to Think With Meta Chain-of-Though. *arXiv preprint arXiv:2501.04682* (2025).

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2305.10601 [cs.CL] https://arxiv.org/abs/2305.10601

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. arXiv:2203.14465 [cs.CL] NeurIPS 2022.