Performative Thinking? The Brittle Correlation Between CoT Length and Problem Complexity

Vardhan Palod[*]
School of Computing and AI
Arizona State University, USA

Karthik Valmeekam
School of Computing and AI
Arizona State University, USA

Kaya Stechly
Dept. of Computer Science and Wu Tsai Institute
Yale University, USA

Subbarao Kambhampati
School of Computing and AI
Arizona State University, USA

Abstract

Intermediate token generation (ITG), where a model produces output before the solution, has been proposed as a method to improve the performance of language models on reasoning tasks. While these reasoning traces or Chain of Thoughts (CoTs) are correlated with performance gains, the mechanisms underlying them remain unclear. A prevailing assumption in the community has been to anthropomorphize these tokens as "thinking", treating longer traces as evidence of higher problem-adaptive computation. In this work, we critically examine whether intermediate token sequence length reflects or correlates with problem difficulty. To do so, we train transformer models from scratch on derivational traces of the A* search algorithm, where the number of operations required to solve a maze problem provides a precise and verifiable measure of problem complexity. We first evaluate the models on trivial free-space problems, finding that even for the simplest tasks, they often produce excessively long reasoning traces and sometimes fail to generate a solution. We then systematically evaluate the model on out-of-distribution problems and find that the intermediate token length and ground truth A* trace length only loosely correlate. We notice that the few cases where correlation appears are those where the problems are closer to the training distribution, suggesting that the effect arises from approximate recall rather than genuine problem-adaptive computation. This suggests that the inherent computational complexity of the problem instance is not a significant factor, but rather its distributional distance from the training data. These results challenge the assumption that intermediate trace generation is adaptive to problem difficulty and caution against interpreting longer sequences in systems like R1 as automatically indicative of "thinking effort".

1 Introduction

Recent advances in general planning and problem solving have been spearheaded by "Long Chain-of-Thought" models, most notably DeepSeek's R1 [8]. These transformer-based large language models undergo the standard stages of pre-training, instruction tuning, and preference alignment, followed by additional post-training on reasoning tasks. At each step, the model is given a question, generates a sequence of intermediate tokens (often called a Chain of Thought or reasoning trace), and ends with a specially delimited answer. After this answer is verified by a formal system, the model's parameters are updated to increase the likelihood of producing correct final outputs.

Although no optimization pressure is typically applied to the intermediate tokens themselves [2, 40], models empirically perform better across many domains when they generate them first [20,

[*] Corresponding author: vpalod@asu.edu

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

30, 38, 10, 7, 8, 22, 19, 15]. While the performance gains are well established, their underlying causes remain unclear. Prior work often interprets these traces anthropomorphically, claiming that models are "thinking" before answering [20, 5, 8, 33, 40, 3]. At the same time, producing longer reasoning traces has been described as *inference-time scaling*—the idea that models perform problem-adaptive computation. Yet there is little reason to expect such adaptivity given how these models are trained. In practice, post-training methods such as supervised fine-tuning with derivational traces or reinforcement learning with rewards based only on final answers are employed [37, 15, 19, 17, 13, 36, 8, 27, 1, 34]. In both cases, models are prompted to generate intermediate tokens before producing their answers, but optimization pressure applies only to the correctness of the final output. The intermediate tokens, being samples from the model's base policy, are not explicitly aligned with problem difficulty, correctness, or structured reasoning. Consequently, these traces are not guaranteed to reflect problem-adaptive computation and may instead be arbitrary byproducts of the generative process.

Following previous work that elucidated important functional aspects of large-scale models through controlled small-scale experiments [29, 23, 39] and working within a sort of "model organism" paradigm, we focus on fully controlled, open, and replicable models trained from scratch. Specifically, we train transformer models from scratch on derivational traces of the A* search algorithm. This approach offers two advantages: (1) the reasoning procedure is explicitly defined and verifiable, and (2) the complexity of input problems can be precisely manipulated.

Within this framework, we evaluate whether intermediate token length in transformer-based models really reflects problem difficulty. We first evaluate the model on the simplest path-finding tasks: free-space problems without obstacles. Solving these should, in principle, require minimal computation, yet we find that the model often performs excessively long computations, and in many cases even fails to produce a solution altogether, thus calling into question the interpretation that the length of intermediate tokens correlates with problem complexity. We then evaluate across diverse maze-generation algorithms. On distributions close to the training distribution, intermediate token lengths show some alignment with problem difficulty. However, this correlation vanishes on more structurally distinct distributions, where trace length and problem complexity become entirely decoupled. These results suggest that the apparent adaptivity of intermediate token length is not evidence of problem-sensitive computation. Instead, it may largely reflect the approximate recall from the training distribution, rather than any genuine alignment between computation length and problem difficulty. Our findings thus suggest that the commonly assumed link between "thinking time" (as measured by reasoning trace length) and task difficulty is misleading.

2 Related Works

Post-Training for Reasoning - Recent progress in improving the reasoning capabilities of Large Language Models (LLMs) has been driven by methods that train models not only on correct answers, but also on reasoning traces that lead to those answers [37, 15, 19, 17, 13, 36, 27, 8, 1, 34]. Whether this is achieved via supervised fine-tuning on trace-augmented datasets or via reinforcement learning techniques like Group Relative Policy Optimization (GRPO), the end result is the same: models "think" for varying amounts of time by outputting additional tokens before outputting their final answers, and performance measures improve. These methods are often paired with claims about how and why they work, which lean on anthropomorphic framings of the intermediate tokens as model "thinking". These claims often treat the length of the intermediate traces as the amount of "thinking" a model does before producing a final answer. Thus, there is an implicit assumption that the traces will be problem-adaptive, i.e., easier problems will have shorter traces and harder problems will have longer traces.

Training Transformers on Traces - Prior efforts have trained models from scratch to mimic search algorithms like A*, Breadth-First Search (BFS), and Monte Carlo Tree Search for tasks in pathfinding, arithmetic, and general problem-solving [14, 6, 32], as well as the DPLL procedure for Boolean SAT problems [21]. However, while these studies train on traces of formal procedures, we believe no other work has analyzed whether there is any correlation between the trained model's traces and the ground truth problem complexity.

3 Methodology

We study a standard grid-based pathfinding domain: finding a legal path from a start to a goal cell in a 30×30 grid where each cell is either free or a wall. The agent begins at the start state and can move up, down, left, or right. The transformer receives a tokenized description of the problem [14, 26] and must output a plan—a sequence of actions that moves the agent only through free cells and terminates at the goal.

Navigation problems are generated using diverse maze-generation algorithms (Appendix A.1), producing varied structural patterns. Following Searchformer and Stream of Search [14, 6], we modify A* to emit a linearized trace: each node creation is logged as create x y cG cH and each expansion as close x y cG cH, where cG is the cost from start ($g(n)$) and cH is the heuristic estimate to goal ($h(n)$). In our experiments, we formalize problem difficulty as the number of operations the A* algorithm takes to solve the problem. Thus, we consider that the problem difficulty increases as the number of operations A* takes to solve the problem. Note that the trace length of A* is directly proportional to the number of operations the algorithm takes to solve the problem. An example of a problem, its A* trace, and solution is given in Appendix A.3.

To construct our training sets, we generate 500,000 mazes using Wilson's algorithm and randomly select a start and goal cell. Then, we use A* with the Manhattan distance heuristic to find an optimal plan for each maze as well as to produce a trace that is saved with each datapoint. We modify the architecture of the Qwen2.5 0.5B [24] to support a vocabulary of exactly 944 different tokens (which reduces the parameter count to about 380 million from 500 million), randomly initialize the model, and then train it for 255,000 training steps with a batch size of 8 on two NVIDIA H100s. Note that we are training empty transformer models from scratch instead of fine-tuning pre-trained models. To assess whether the model has learned to solve pathfinding problems, we first evaluate it on 1,000 held-out Wilson mazes, where it achieves 80% accuracy in generating correct plans. We also evaluate this model on held-out problems generated by distinct maze-generation algorithms: Free-space, Kruskal, DFS, SF-Style, and Drunkard. Additional training details are given in Appendix A.2.

4 Results

4.1 Excessive Intermediate token length for the easiest problems

To probe the model's behavior on the simplest possible tasks, we evaluated it on 100 *free-space* problems, where the grid contains no obstacles and the agent must simply navigate from the start state to the goal. More details on the generation of these problems are provided in the Appendix.

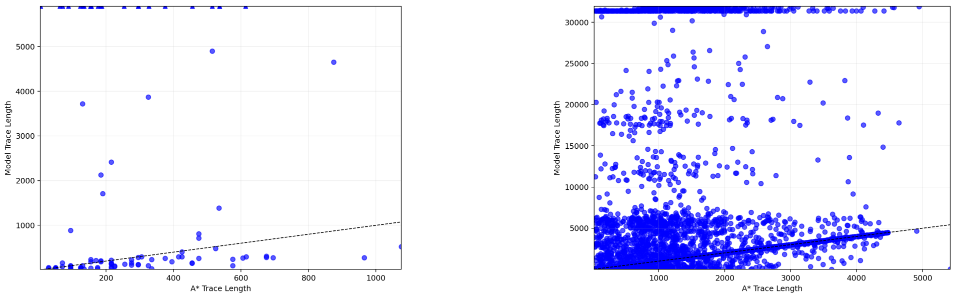
Even under this trivial setting, the model performs poorly: only 5 out of 100 problems yield a valid plan. More strikingly, as shown in Figure 1a, there is very little correlation between the model's intermediate token length and the ground-truth A* trace length, as almost no responses lie on or near the y = x line. If there were a correlation, the generated trace lengths would closely match the ground-truth values. In fact, on some of the easiest problems, the model continues producing tokens up to the maximum context limit of 32k without ever producing a valid solution.

4.2 Analysis of responses on problems generated by various algorithms

We further evaluate the model, trained on Wilson-generated problems, on held-out instances produced by five distinct maze-generation algorithms: Wilson, Kruskal, DFS, SF-Style, and Drunkard. As shown in Figure 1b, the core finding persists: the amount of computation, as reflected in the length of intermediate traces, remains largely agnostic to problem complexity. The scatter plots are broadly populated across the range of problem difficulties, indicating only a very loose correlation between the model's intermediate trace lengths and the ground-truth A* trace lengths.

Analyzing In-Distribution vs. Out-of-Distribution Responses

We also evaluate differences in the model's behavior on held-out in-distribution problems compared to out-of-distribution problems. Since the model is trained on Wilson-generated mazes, it learns to approximate A* traces in those cases, yielding a visible alignment between intermediate token lengths and ground-truth trace lengths (Fig. 2a). However, when evaluated on Searchformer-style mazes [14], this correlation disappears entirely (Fig. 2b). In such cases, the intermediate trace

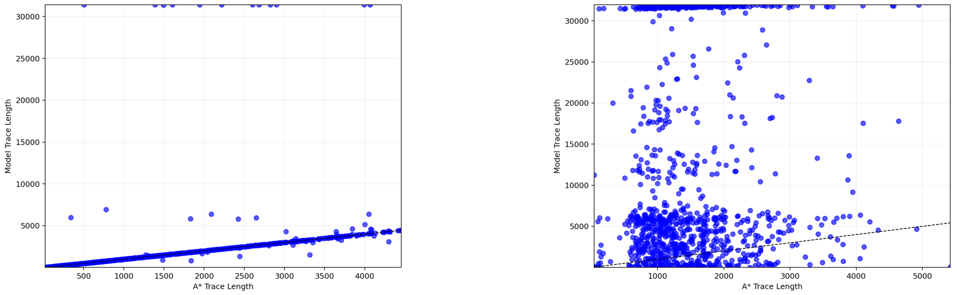


(a) Responses on *Free-space problems*. Points at the top correspond to failures where the model generated tokens until reaching the 32k limit.

(b) Responses on problems generated with DFS, Drunkard, Searchformer-style, Wilson, and Kruskal algorithms (1000 each).

Figure 1: Comparison of generated intermediate token length (Y-axis) with ground-truth A* trace length (X-axis). The dashed line represents $y = x$.

lengths fluctuate independently of problem complexity. Together, these results suggest that the apparent problem-adaptive computation observed under Wilson mazes is due to the test set problem instances, even though held out, being drawn from the same distribution as the training data. As the distributional distance increases—as in the case of Searchformer-style mazes—the correlation between trace length and problem complexity vanishes.



(a) Wilson trace length scatter

(b) Searchformer trace length scatter

Figure 2: Comparison of trace length scatter plots for problems generated using Wilson and Searchformer Algorithms.

5 Discussion

While intermediate token generation has often been interpreted as reflecting problem-adaptive computation, our results suggest that this assumption is misleading. When the distributional distance between test and training data is small, as in the case of Wilson mazes, intermediate token length may appear correlated with problem complexity. However, as this distance increases, such as with Searchformer-style or Drunkard mazes, the correlation vanishes. These observations carry implications for how reasoning tokens are interpreted. Broadly, our results indicate that trace length is not a function of the from-scratch computational complexity of the problem instance, but rather of the distributional distance between that instance and the instances that the LLM/LRM has been trained on. Since intermediate token generation does not reliably indicate "thinking" or the computational effort expended to solve a problem, methods that claim to improve efficiency in large reasoning models by reducing reasoning-chain length may be based on a flawed premise [35, 1, 4, 9, 18, 16, 25].

6 Conclusion

In this work, we analyzed whether the length of intermediate tokens reflects problem-adaptive computation in transformer-based large reasoning models (LRMs). Using a controlled setting with transformers trained from scratch on verifiable A* traces, we demonstrated that intermediate token length and problem complexity are loosely correlated.

Our findings suggest that the commonly assumed link between "thinking time" (as measured by reasoning trace length) and task difficulty is misleading. More broadly, our results caution against anthropomorphizing intermediate reasoning tokens and highlight the need for more rigorous methodologies when interpreting the internal processes of transformer-based reasoning models.

Acknowledgment

This research is supported in part by grants from ONR (N00014-25-1-2301 and N00014-23-1-2409), DARPA (HR00112520016), DoD RAI (via CMU subcontract 25-00306-SUB-000), an Amazon Research Award, and a generous gift from Qualcomm.

References

[1] Daman Arora and Andrea Zanette. Training language models to reason efficiently, 2025. *URL https://arxiv. org/abs/2502.04463*.

[2] Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.

[3] Sébastien Bubeck, Varun Chadrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[4] Zigeng Chen, Xinyin Ma, Gongfan Fang, Ruonan Yu, and Xinchao Wang. Verithinker: Learning to verify makes reasoning model efficient. *ArXiv*, abs/2505.17941, 2025.

[5] Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.

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

[7] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[9] Michael Hassid, Gabriele Synnaeve, Yossi Adi, and Roy Schwartz. Don't overthink it. preferring shorter thinking chains for improved llm reasoning. *ArXiv*, abs/2505.17813, 2025.

[10] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

[11] jrheard. Procedural dungeon generation: A drunkard's walk in clojurescript.

[12] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[13] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[14] Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

[15] Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, et al. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025.

[16] Ruosen Li, Ziming Luo, Quan Zhang, Ruochen Li, Ben Zhou, Ali Payani, and Xinya Du. Aalc: Large language model efficient reasoning via adaptive accuracy-length control. *ArXiv*, abs/2506.20160, 2025.

[17] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[18] Yule Liu, Jingyi Zheng, Zhen Sun, Zifan Peng, Wenhan Dong, Zeyang Sha, Shiwen Cui, Weiqiang Wang, and Xinlei He. Thought manipulation: External thought can be efficient for large reasoning models. *ArXiv*, abs/2504.13626, 2025.

[19] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[20] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. 2021.

[21] Leyan Pan, Vijay Ganesh, Jacob Abernethy, Chris Esposo, and Wenke Lee. Can transformers reason logically? a study in sat solving. *arXiv preprint arXiv:2410.07432*, 2024.

[22] Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

[23] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[24] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[25] Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025.

[26] DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. In *The Thirteenth International Conference on Learning Representations*, 2024.

[27] Hao Sun. Reinforcement learning in the era of llms: What is essential? what is needed? an rl perspective on rlhf, prompting, and beyond. *arXiv preprint arXiv:2310.06147*, 2023.

[28] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[29] Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024.

[30] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[31] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 296–303, 1996.

[32] Mengjiao Sherry Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. *Advances in Neural Information Processing Systems*, 35:36366–36381, 2022.

[33] Shu Yang, Junchao Wu, Xin Chen, Yunze Xiao, Xinyi Yang, Derek F Wong, and Di Wang. Understanding aha moments: from external observations to internal mechanisms. *arXiv preprint arXiv:2504.02956*, 2025.

[34] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[35] Danlong Yuan, Tian Xie, Shaohan Huang, Zhuocheng Gong, Huishuai Zhang, Chong Luo, Furu Wei, and Dongyan Zhao. Efficient rl training for reasoning models via length-aware optimization. *ArXiv*, abs/2505.12284, 2025.

[36] Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

[37] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[38] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[39] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in neural information processing systems*, 36:27223–27250, 2023.

[40] Hengguang Zhou, Xirui Li, Ruochen Wang, Minhao Cheng, Tianyi Zhou, and Cho-Jui Hsieh. R1-zero's" aha moment" in visual reasoning on a 2b non-sft model. *arXiv preprint arXiv:2503.05132*, 2025.

A Appendix

A.1 Maze Generation Algorithms

We generate navigation problems using diverse generation algorithms, resulting in varied structural patterns and exploration dynamics. This enables systematic out-of-distribution (OOD) evaluation by testing models on maze types unseen during training – which was all done on mazes generated with Wilson's algorithm. These generation algorithms can be sorted into two major categories: 1) algorithms that do not permit cycles and sample over the spanning trees of the 30×30 grid, and 2) algorithms that permit loops and create noisy, less-structured dungeon or cave-like instances. For all algorithms except Searchformer's, which has its own start and goal generation loop, we sample a legal (start, goal) pair after maze generation. We also generate problems with no obstacles in the grid called free-space problems.

Acyclic Maze Generation

1. **Wilson's algorithm:** This is the algorithm that we use to generate mazes for training models. Wilson's algorithm generates uniform random mazes by performing loop-erased random walks from unvisited cells until they connect to the current maze [31]. Each walk removes any loops it creates, ensuring a valid tree structure. This process continues until all cells are included, producing a uniform sample from the space of all possible spanning trees of the 30×30 graph.
2. **Kruskal's algorithm:** Kruskal's algorithm, originally proposed for finding a minimum spanning forest of an undirected edge-weighted graph [12], generates mazes by treating each cell as a node and randomly removing walls between unconnected regions, using a union–find structure to avoid cycles. This results in a fully connected maze without loops, though the maze distribution is not perfectly uniform. The method produces mazes biased towards short local connections and dead ends.
3. **Randomized Depth-First Search algorithm:** The randomized depth-first search (DFS) or recursive backtracker algorithm generates mazes by carving a path forward until reaching a dead-end [28]. When it hits a dead-end (no unvisited neighbors), it backtracks until it finds a new direction to explore, repeating until all cells are visited and connected into a complete maze. Depth-first search is biased towards generating mazes with low branching factors and many long corridors.

Cave Generation

4. **Drunkard's Walk:** We implement a version of the "Drunkard's Walk" algorithm, as described by [11], and originally used for procedurally generating dungeons for top-down two-dimensional video games. Starting from a grid of solid walls, a random walk is performed, carving out the current cell on every step. The walk continues until a predefined number or percentage of floor tiles has been dug out. This method preserves cycles, producing cave-like structures with open chambers and looping corridors. The output space includes grid states unreachable by perfect acyclic maze generators.
5. **Searchformer style generation:** We also implement the random generation algorithm used in the Searchformer paper [14], though we use it for evaluation rather than training. Tasks are generated by exhaustive rejection sampling: first, randomly select a number between 30% and 50%. Then select that percentage of cells to be wall cells. Randomly choose a start and goal location and execute A* to find an optimal plan. Reject unsolvable, too easy, or duplicate instances and resample. These instances also allow for loops and so are also out of distribution for our models.

No-Obstacles Generation

6. **Free-space algorithm:** In this algorithm, we first select the number of levels of outer walls. In our case, we chose the number of levels of outer walls to be 4. Within the inner grid, a start cell and a goal cell are then chosen randomly, with no walls placed in the inner grid. Finally, we execute A* to generate the intermediate trace and obtain the optimal plan.

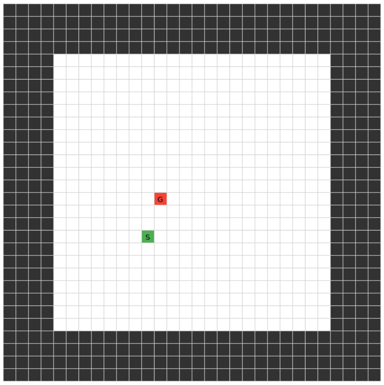


Figure 3: Example grid of a Free-space problem. It is a 30x30 grid with no obstacles in the inner grid. The start and goal states are chosen randomly.

A.2 Model Training and experimental details

For all experiments, we trained decoder-only Qwen-2.5-0.5B models, customized with a domain-specific tokenizer that reduced the effective parameter count to approximately 380M. The models were randomly initialized and trained for 255,000 steps with an effective batch size of 8 on two NVIDIA H100 GPUs. The model has a context length of 32,000 tokens to support the long lengths of intermediate token generation. Training was performed on a dataset of 500k Wilson generated problems, after which the models were evaluated on 1,000 held-out instances generated by five distinct maze generation algorithms: Wilson, Kruskal, DFS, SF-Style, and Drunkard.

We optimized with AdamW (β_1=0.9, β_2=0.999) and applied a weight decay of 0.1528, a peak learning rate of 2.2758e-4, and 100 warm-up steps. All experiments were run under bf16 precision with fixed random seeds for reproducibility.

A.3 Example Problem Instance

Problem description, A* trace, and solution are given in tokenized form, following [14, 6]. The visualization of the problem is given in figure 3.

Problem

```
start 18 11 goal 15 12 wall 0 0 wall 0 1 wall 0 2 wall 0 3 wall 0 4 wall 0 5
wall 0 6 wall 0 7 wall 0 8 wall 0 9 wall 0 10 wall 0 11 wall 0 12 wall 0 13 ..
```

A* search trace

```
close 18 11 c0 c4 create 17 11 c1 c3 create 19 11 c1 c5 create 18 10 c1 c5
create 18 12 c1 c3 close 17 11 c1 c3 create 16 11 c2 c2 create 17 10 c2 c4
create 17 12 c2 c2 close 18 12 c1 c3 create 19 12 c2 c4 create 18 13 c2 c4
close 16 11 c2 c2 create 15 11 c3 c1 create 16 10 c3 c3 create 16 12 c3 c1
close 17 12 c2 c2 create 17 13 c3 c3 close 15 11 c3 c1 create 14 11 c4 c2
create 15 10 c4 c2 create 15 12 c4 c0 close 16 12 c3 c1 create 16 13 c4 c2
close 15 12 c4 c0
```

Solution

```
plan 18 11 plan 17 11 plan 16 11 plan 15 11 plan 15 12
```