IS IN-CONTEXT SEARCH WORTH ITS COMPUTE?

Anonymous authors
Paper under double-blind review

ABSTRACT

Training language models on search tasks can be substantially improved by providing supervision via search-augmented (SA) intermediate traces that imitate sequential search. Yet, this requires processing longer sequences than solution-only (SO) training, and prior results leave it unclear when this additional supervision is compute-efficient. We therefore ask: when is learning to imitate search more compute-efficient than learning solutions directly? We study this question on maze navigation and Countdown by systematically varying three factors: training scale, the structure of the underlying search problem, and the structure of the intermediate supervision. We identify two different scaling regimes. On maze navigation, SO supervision reaches low error with less training compute than search imitation; we term this a *search-optional* regime. On regular Countdown, SA supervision with depth-first search traces is substantially more compute-efficient than SO across the training budgets we evaluate; we term this a *search-efficient* regime. The advantage of search traces increases on modular Countdown, where the magnitude-based heuristics perform at chance level. Finally, training on incomplete search records reduces the efficiency advantage of SA learning, while semantically unrelated traces behave differently across tasks. These results show that the relative efficiency of SA and SO supervision depends jointly on training scale, problem structure, and supervision quality. Exploiting this insight, we show that selective SA supervision achieves low error with less training compute in joint training on both tasks than uniformly applying either SA or SO supervision.

1 INTRODUCTION

Large language models are becoming increasingly better at solving problems in mathematics, coding, and planning that require sequential reasoning steps (DeepSeek-AI et al., 2025; Team et al., 2026; Yang et al., 2025a). A promising approach for improving this capability is to train with *search-augmented* (SA) Chain-of-Thought (CoT), where a model learns to imitate a *structured* sequential search process that explores problem states systematically, with state expansions, subtree evaluations and backtracking, until a solution is found (Xiang et al., 2025; Kim et al., 2025; Lehnert et al., 2024; Gandhi et al., 2024; 2025). However, training SA models requires search-trace annotations, which substantially increase training- and test-time compute and may require a separate data-collection effort. Their utility, therefore, needs to be compared against *solution-only* (SO) models (Lehnert et al., 2024) that generate solution steps directly without exploring alternative paths, thereby not incurring additional costs imposed by search traces.

Existing work on algorithmic search problems generally finds that SA models outperform SO models (Gandhi et al., 2024; Lehnert et al., 2024; Valmeekam et al., 2026; Qin et al., 2025). However, SO models can recover SA performance when trained on more data (Lehnert et al., 2024), while several comparisons use small fixed datasets, potentially under-training the models (Qin et al., 2025; Valmeekam et al., 2026). Moreover, models can retain much of the benefit of search traces even when those traces are semantically unrelated to the problem (Valmeekam et al., 2026). It therefore remains unclear whether the advantage of search imitation persists at training compute parity, how it depends on the problem, and whether it comes from search semantics or additional serial in-context computation. This motivates a systematic comparison along three axes: *training scale*, *the structure of the search problem*, and *the structure of the intermediate supervision*, to answer the following:

When is learning to imitate search more compute-efficient than learning solutions directly?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We answer this question using maze navigation from Lehnert et al. (2024) and two variants of Countdown[1]: a regular version adapted from Gandhi et al. (2024), and a modular version (ours). Both tasks are graph search problems but differ in structure, representation, and heuristics. Across these tasks, we vary dataset and model sizes and compare SO supervision with search traces and semantically unrelated traces. On Countdown, we also compare Stream-of-Search (SoS) (Gandhi et al., 2024) and depth-first search (DFS) traces on both variants.

We find that SA and SO models exhibit different scaling behaviours across tasks and intermediate traces. On maze navigation, SO models use less training compute to reach the same near-perfect performance as SA models with optimal A* traces, despite requiring more training samples. Increasing model size further reduces this sample-efficiency gap. In contrast, on Countdown, SA models with DFS traces retain a substantial compute-efficiency advantage across the training budgets we study. We refer to these scaling regimes as *search-optional* and *search-efficient*, respectively.

Moving from regular to modular Countdown increases the efficiency gap between SA-DFS and SO models, consistent with the loss of useful magnitude-based heuristics making SO learning harder. The choice of intermediate supervision also matters. On regular Countdown, SA models trained on sub-optimal SoS records are less compute-efficient than SO models. Unrelated traces outperform SO models in sample efficiency on modular Countdown but underperform them on maze navigation; genuine search traces outperform unrelated traces on both tasks. Together, these results show that the relative efficiency of SA and SO supervision depends on training scale, the underlying search problem, and intermediate supervision.

Finally, we argue that, since search traces consume additional training tokens, uniformly applying SA supervision may be inefficient when its benefit varies across tasks. We test this by jointly training on maze navigation and modular Countdown. Our results show that selectively using SO supervision for maze navigation and SA supervision for modular Countdown reaches low error on both tasks with less training compute than uniformly using either SO or SA supervision.

Our contributions can therefore be summarised as follows:

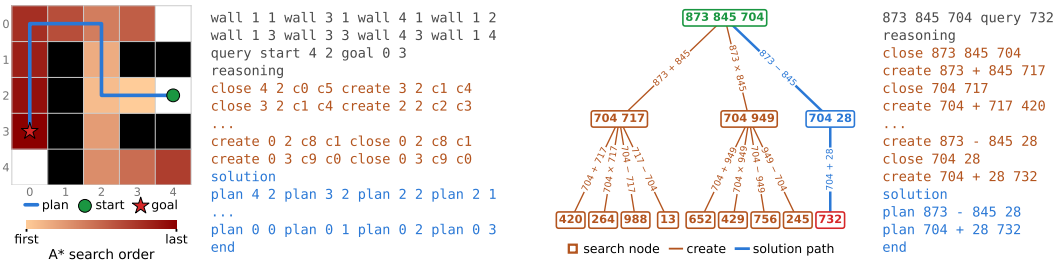
- We systematically characterise the relative efficiency of search-augmented (SA) and solution-only (SO) supervision along three axes: training scale, the structure of the underlying search problem, and the structure of the intermediate supervision.
- We identify two qualitatively different scaling regimes, *search-optional* and *search-efficient*. On maze navigation, SO supervision reaches low error with less training compute. On Countdown, SA supervision with DFS traces retains a compute-efficiency advantage across the training budgets we evaluate.
- We show how changing the supervision or the problem structure can substantially alter the relative advantage of SA training.
- We show that selectively allocating SA supervision across a mixture of tasks can improve performance per unit of compute compared with applying either SA or SO supervision uniformly.

2 Related work

Search with language models. Language models paired with an external search process that uses them for action generation and state evaluation (Yao et al., 2023; Hao et al., 2023) gain substantial improvements over language models that attempt to decode optimal solution steps directly (Wei et al., 2022; Nye et al., 2022). However, reliance on a fixed external algorithm to perform search limits the flexibility of this approach. In response, pretraining with sequential search in-context was proposed to allow models to manage the search process themselves and potentially discover more efficient search strategies (Gandhi et al., 2024; Lehnert et al., 2024; Yang et al., 2022; Schultz et al., 2025), with follow-up studies exploring if models can learn to switch solution strategy dynamically (Su et al., 2025), use pointers to track progress (Kang et al., 2026), and move learning to search to the post-training stage (Opedal et al., 2026; Kim et al., 2025).

We build on this prior work and contribute an evaluation of compute efficiency and scaling behaviour of in-context search, which shows that its advantage over direct solution decoding is not universal.

[1] https://en.wikipedia.org/wiki/Countdown_(game_show)



(a) Maze navigation.

(b) Countdown.

Figure 1: Examples of search problems used in this study and their training representation. SA models generate an intermediate search trace followed by a solution; SO models generate only the solution. (a) In maze navigation, the solution is a path from a start cell to a goal cell. The intermediate trace is a serialised execution trace of the A* algorithm. (b) In Countdown, the solution is a sequence of arithmetic operations that produces the target. The example shows a variant with modular arithmetic. The intermediate trace is a serialised DFS algorithm.

This complements findings that extended reasoning can sometimes be unnecessary or harmful (Liu et al., 2025; Kurtulush, 2026). Specifically, we use a maze navigation task from Lehnert et al. (2024) and a Countdown task from Gandhi et al. (2024), but scale their setup to much larger datasets. Other studies evaluate other aspects of SA models: Qin et al. (2025) compare them to SO models at test-time compute parity, and Valmeekam et al. (2026) investigate SA with traces semantically unrelated to the task. However, both studies use small fixed-size datasets without adjusting compute or investigating their scaling behaviour; indeed, our results suggest that training on the same tasks but with more data and evaluating models at training compute parity reverses some of these works' conclusions.

Learnability and intermediate computation. Analyses of transformers on graph problems establish the conditions under which SO models can fail. On pathfinding tasks, a transformer must learn a path-merging algorithm in its forward pass to decode each next node correctly (Saparov et al., 2025), but previous works show that the model may develop heuristic-based shortcut solutions if the problem's in-context graph diameter exceeds the model's capacity (Ye et al., 2026). Learning tasks with hard initial tokens and easy subsequent ones, for example, path-star graphs, can fail regardless of data amount (Bachmann & Nagarajan, 2024). A separate line of theoretical work demonstrates that extending computation with intermediate CoT traces improves sample efficiency (Wen et al., 2025; Abbe et al., 2024; Yang et al., 2025b;c). Our results are consistent with this literature, to which we contribute additional controlled experiments on non-graph tasks and empirical results that support theoretical claims in a more realistic setup.

3 Experiment Setup

Maze navigation. We adopt a variation of the maze navigation task introduced by Lehnert et al. (2024). Here, a maze is a 2D grid where each cell is either a wall or a path, specified using a row-wise sequence of wall coordinates (Figure 1a). We place the query (i.e., the coordinates of the start and the goal cells) after the maze specification to prevent the maze description embeddings from adapting to the given query. When training SA models, we generate training search traces representing serialised step-by-step executions of the A* algorithm (Hart et al., 1968). These traces operate on two sets: a "frontier" set of path cells (i.e., the current leaves of the search tree) that have not yet been expanded, and a "closed" set of cells that have been expanded and whose children have been added to the frontier. Transitions between cells are implicit, and we record only the resulting states. For both SO and SA models, the solution is a sequence of path cells from the start to the goal. We generate 3 million mazes using Wilson's algorithm (Wilson, 1996) and select one start-goal cell pair for each maze at random.

Countdown. The objective of Countdown is to derive an arithmetic expression from a list of integers that evaluates to a target; the puzzle can be viewed as a walk over a graph whose nodes are integer multisets of decreasing size (Figure 1b). We experiment with two variants of the game. The first

variant is derived from Gandhi et al. (2024), which uses regular arithmetic, a multiset of 4 integers drawn from $[1, 99]$, and a target integer $t \in [10, 100]$. We constrain all intermediate results to the range $[0, 999]$ to prevent under-training of the rare large intermediate values, which results in the removal of 4.78% of the puzzles from the dataset compared to the original. The second is our variant with modular arithmetic. It contains a multiset of 3 integers drawn from $[0, 1000]$ and a target integer $t \in [0, 1000]$. We allow addition $(+)$, subtraction $(-)$, and multiplication $(\times)$ modulo 1001. To control puzzle complexity, we impose three constraints: (1) the solution uses all N input numbers, (2) every puzzle in the dataset has the same length N.

For each variant, we generate two types of traces. The first is an SoS trace, which is a mixture of search strategies with different heuristics introduced by Gandhi et al. (2024). This strategy prunes the search tree such that only 59.8% of the traces find the target on regular Countdown and 54.6% of the traces on our modular Countdown. The second is a DFS trace, generated by running the depth-first search (DFS) algorithm until a solution is found, resulting in a 100% success rate on both variants. Unsuccessful SoS searches end with an empty solution, whereas every DFS record contains a successful solution. The comparison therefore changes both the search trace and the availability of solution supervision. A combination of two trace types and two Countdown variants results in four datasets, with 20 million puzzles in each.

Models and training setup. Throughout this section, we investigate our research questions by comparing SA and SO models on search problems as data and model parameters increase. In each scaling experiment, we train GPT-2 models from scratch with 5 seeds, except for the mixture experiments, which use one run per configuration. Models are trained either with search traces (SA) or without (SO) before they generate a solution. We scale data by subsampling overlapping fractions of the full dataset of increasing size, and scale models by increasing the number of layers and/or attention heads. Each run completes its full training cosine schedule, since early stopping in longer runs biases the loss upwards (Hoffmann et al., 2022). Every evaluation dataset consists of 1,000 held-out examples. We prompt SO and SA models with a puzzle specification and generate solutions with greedy decoding.

We provide more details about the tasks in Appendix A and training setup in Appendix B.

4 When is Search-Augmented Supervision Worth It?

Our goal for these experiments is to investigate empirically when search trace supervision is more compute-efficient than SO supervision, and why this might be the case. To do this, we compare models with both supervision formats along the three axes: training scale, the structure of the underlying search problem, and the structure of the intermediate supervision. Thus, for each axis, we formulate a research question:

RQ1. How does the compute efficiency of SA and SO models compare as training is scaled along dataset and model sizes?

RQ2. How do changes to the search problem affect learning efficiency with SA supervision?

RQ3. How do changes to the search trace affect the efficiency of learning with SA supervision?

4.1 SO models outperform SA models on maze navigation

We start by investigating the finding by Lehnert et al. (2024) that SO models can reach the performance of SA models with larger and more diverse datasets. Given that SA models require more compute for training and inference, we characterise the trade-off between sample efficiency and compute requirements more precisely (RQ1). In these experiments, we train transformers with 4 to 12 layers and 12 heads (GPT-2-Small) on fractions of a full dataset of 3 million mazes for 4 epochs, resulting in a maximum training budget of 12 million samples. See Appendix A.1 for details and examples.

Search trace supervision becomes unnecessary on maze navigation with scale (RQ1; Figure 2). Our results show that, for the maze navigation setup of Lehnert et al. (2024), (1) increasing dataset size allows SO models to generalise with less training compute compared to SA models, at any model size, and (2) increasing model size improves sample efficiency of both SO and SA models,

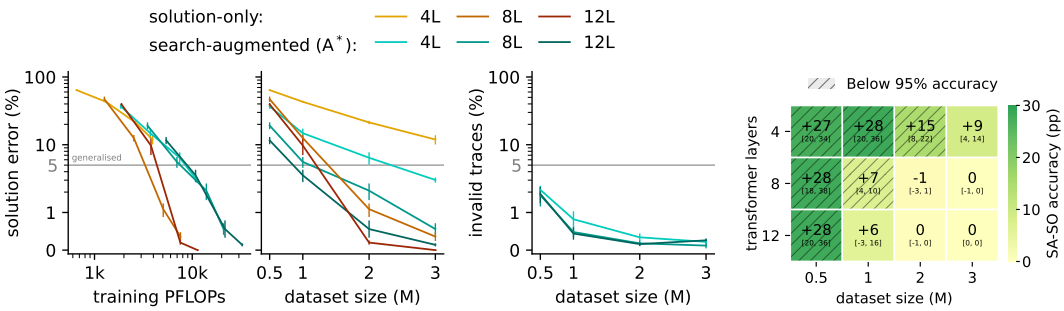


Figure 2: Maze navigation is *search-optional*: search traces help only when models are under-trained, and their advantage disappears with scale. (a) SO models reach low solution error with less training compute. Increasing model size improves the sample efficiency of both SA and SO models. Values are arithmetic means over 5 seeds, with error bars showing one standard error of the mean. (b) SA models are more accurate only when both SO and SA models are under-trained (shaded), and this advantage disappears with scale. Values are arithmetic means with Student-t 95% intervals over 5 seeds.

although SO models improve faster (Figure 2a, solution error). Overall, the sample efficiency advantage of SA models disappears once both supervision approaches generalise in-distribution (Figure 2b). This results in a superior use of resources by the SO models: for example, the 12-layer SO model reaches 5% solution error with the same amount of training data (approx. 1.25M) as the 8-layer SA model, while using 2x less FLOPs.

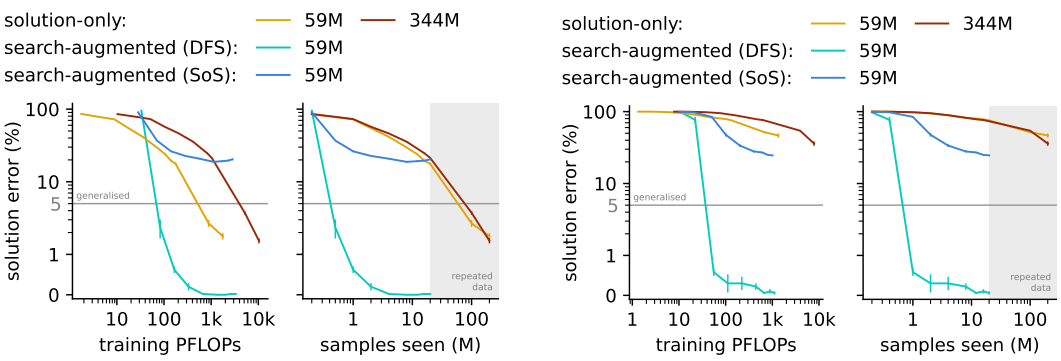
One obvious explanation for the poor compute efficiency of SA learning is the additional sequence length. For example, search traces in our maze search problem contain 10 times more tokens on average than solutions themselves. However, trace length alone does not explain the result, since SA models learn to produce valid search traces efficiently. Instead, they seem to have difficulty extracting solutions from the traces. For instance, the 4-layer SA model trained on 0.5M mazes reaches a 2% invalid trace rate while still having a 40% solution error (Figure 2a, solution error and invalid traces). This points to a non-obvious bottleneck: on maze navigation, learning to generate a solution path remains the main difficulty for transformers, even when a valid search trace that contains the exact solution is within the model's context. Our results suggest that, for maze navigation problems, adding search traces has limited impact on performance but a substantial negative impact on training and test-time compute efficiency.

4.2 SA models with DFS traces outperform SO models on Countdown

In the previous section, we showed that SO models can generalise with less compute than SA models as model size becomes sufficiently large. However, we hypothesise that this is not the case for all search problems. The coordinate-based maze representation has several properties that transformers can exploit: (1) the full state-space graph is present in context and can be attended to during solution generation; (2) the state transition function is local (i.e., the next state in the solution is always adjacent to the previous state); and (3) Manhattan distance provides a useful heuristic for selecting states on the optimal path. These properties may make the maze navigation task relatively easy to learn without search traces.

To test the effects of scaling on problems without these properties, we use two variants of the Countdown game. The representation of Gandhi et al. (2024) Countdown puzzles does not contain the full state graph in context, and its state transition function is non-local; modular Countdown additionally removes reliance on magnitude-based heuristics. We scale each variant across dataset and model sizes to compare the compute efficiency of SO and SA models at training compute parity (RQ1), then we test the effects of changing the problem structure without changing the underlying task (RQ2), and, finally, the effects of changing the search trace (RQ3).

For each variant of the game, we train 8-layer transformers with 12 heads and 59 million parameters on fractions of a full dataset with 20 million puzzles for 1 epoch. We also probe the limits of



(a) Gandhi et al. (2024) Countdown

(b) Modular Countdown (ours)

Figure 3: Countdown with DFS traces is *search-efficient*: the advantage of search traces persists across the scales we study. On both (a) the Countdown from Gandhi et al. (2024) and (b) our modular Countdown, SA-DFS models reach low solution error with less data and training compute. Increasing model size does *not* improve the sample efficiency of SO models, which fail to generalise on modular Countdown even at substantially larger scale. By contrast, sub-optimal SoS traces make SA models less efficient than SO models on the Countdown from Gandhi et al. (2024), placing this configuration in the search-optional regime. Values are arithmetic means over 5 seeds, with error bars showing one standard error of the mean.

SO training by repeating each Countdown dataset for up to 10 epochs, resulting in 200 million seen examples, and by scaling the model size to a 12-layer, 24-head transformer with 344 million parameters and double the model width. We provide more details about problem structure and training setup in Appendix A.2.

Search trace supervision is more efficient on Countdown at *any* scale (RQ1; Figure 3). Our results on regular Countdown from Gandhi et al. (2024) show that SA supervision with DFS traces is the most compute- and sample-efficient approach to learning the task, at any model or dataset scale (Figure 3a, DFS trace). Specifically, SO models generalise only after seeing close to 80 million examples, while SA-DFS models only need around 400k samples, or 200x less data and 7x less compute. This result demonstrates a qualitatively different scaling behaviour of SA and SO models to that on maze navigation: (1) SO models take a lot more data to generalise than SA models, using more training compute, and (2) increasing model size does not change sample efficiency of SO models, and only increases training compute.

Although SO models underperform SA models on Gandhi et al. (2024) Countdown, they can still use heuristics as a shortcut to make progress on learning the task without search. Specifically, in the first expression, preferring the smallest result succeeds on 43% of the puzzles and preferring the result closest to the target succeeds on 32% of the puzzles, while choosing the first result at random succeeds only 23% of the time. Also, the first operation is either $(+)$ or $(-)$ in 86% of the puzzles. On modular Countdown, the two tested magnitude-based heuristics perform at chance level, and first-operation frequencies are more balanced. We provide a detailed comparison of Countdown heuristics in Appendix A.3. Besides diminishing the effect of heuristics, we also increase the input range to cover the entire vocabulary. At the same time, we simplify other aspects of the puzzle compared to the regular version, by using sorted input, shorter puzzles, and excluding the $(\div)$ operation. As a result, modular Countdown is easier to solve using search but harder to solve using heuristics and memorisation.

Removing heuristics from Countdown hurts SO models but not SA models (RQ2; Figure 3). Our results show that, on modular Countdown, SO models fail to generalise even with 20x more unique data, 200x more seen data in training, and 25x the training compute of the most compute-efficient generalised SA model, still failing on 55% of held-out puzzles (Figure 3b, 59M). Increasing model size to 344M does not change the sample efficiency of SO models until the largest sample size of 200M puzzles, where the model's error drops to 33% (Figure 3b, 344M). Thus, removing useful heuristics costs the best SO model about 21x more data to achieve 33% error. Furthermore, it is not clear what dataset or model size would be required for SO models to generalise on this task,

since their error curves do not show the predictable trajectory with FLOPs or data size observed for maze navigation.

At the same time, SA-DFS models slightly improve their sample and compute efficiency on the modular variant. This suggests that the presence or absence of heuristics does not have any impact on the solution learned by the SA models, since they also do not impact DFS traces. Instead, SA models are impacted by the trace length: switching from a regular variant with 4 integers to the modular variant with 3 integers reduces the maximum search tree width from 280 nodes to 48 nodes, and the maximum depth from 3 to 2. This reduces the length of the DFS traces (for details, see Table 2 in Appendix A.4), which results in faster and more efficient learning. Conversely, the efficiency of SO models does not benefit from a smaller search tree, suggesting they do not learn to search.

4.3 Sub-optimal and unrelated search traces hurt efficiency

In a realistic setting, optimal search traces may be intractable or hard to obtain. Gandhi et al. (2024) developed their version of Countdown to demonstrate that SA models with sub-optimal SoS traces still outperform SO models. However, the models were trained on a fixed dataset of 500,000 puzzles. To test the performance of SoS traces relative to optimal DFS traces and SO supervision at scale, we train them on both Countdown variants with up to 20M samples.

Search supervision with sub-optimal traces hurts efficiency (RQ3; Figure 3, SoS trace). Our results show that using sub-optimal traces harms learning: the models reach at most a 19% solution error rate. This allows SO models to overtake SA-SoS models in compute efficiency and accuracy. Specifically, SO models of the same size reach the minimum error achieved by SA-SoS models with 2.16x more data but 7.4x less compute; afterwards, they proceed to improve, while the accuracy of SA-SoS models plateaus. The same trend seems to appear on modular Countdown, although SO models there are less compute-efficient than SA-SoS models due to the difficulty of learning the task; the scaling curves suggest that with more data, SO models will nevertheless be able to outperform SA-SoS models. The results suggest that sub-optimal traces benefit learning only on tasks which are challenging for SO models, and only up to a certain dataset size; at scale, training with sub-optimal traces can be detrimental to performance.

Another way to avoid relying on optimal search traces is to replace them with alternative intermediate traces which may be cheaper to obtain. A particular example is the traces whose content is unrelated to a given task; thus, their tokens only provide extra in-context computation. Valmeekam et al. (2026) investigated this on maze navigation by replacing the $A^\star$ trace in each sample with a trace from a different, unrelated sample. They showed that models trained on this augmented dataset achieved the same performance as SA models trained with the original, related traces. However, as demonstrated in Section 4.1, this maze navigation task is search-optional, and search traces are not required to learn it efficiently. To test whether their result translates to our training setup, we train an 8-layer transformer with swapped traces on the maze navigation setup from Section 4.1 and on the modular Countdown setup from Section 4.2.

Unrelated traces underperform optimal traces on all tasks (RQ3; Figure 4). We find that unrelated traces perform worse than SA models with optimal traces at the same data or compute budget. This is not surprising, given the results in previous sections. On maze navigation, SO models generalise efficiently without traces. Thus, it is possible that the models learn to ignore such a trace, which still allows them to perform well. Adding unrelated traces, therefore, adds compute without any benefit, while being sensitive to hyperparameters could explain the discrepancy with prior results. On modular Countdown, search traces are important for efficient generalisation: our experiments showed that without the traces, models rely on heuristics to solve the problem, and not search. It is, therefore, unlikely that tuning hyperparameters can make unrelated traces perform on par with genuine search traces.

Unrelated traces can outperform SO models on some tasks (RQ3; Figure 4, countdown). Although unrelated traces underperform search traces, our results show that, on modular Countdown, unrelated traces have a clear sample-efficiency advantage over SO models, achieving a 19% solution error rate with 20M samples against a 75% error rate of SO models. This is surprising, since unrelated traces contain no information about a given task instance, although every trace contains many examples of modular arithmetic expressions. This suggests that using intermediate traces to solve unrelated problems, which involve 'skills' that *are* related to the original problem, can improve

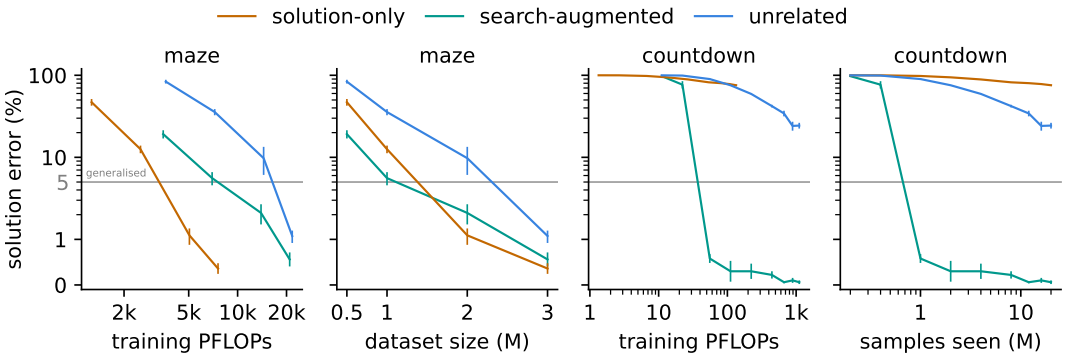


Figure 4: Unrelated traces are consistently less effective than genuine search traces across all tasks and scales, contrary to prior results. On modular Countdown, however, they are more sample-efficient than SO models. Unrelated trace datasets are constructed by swapping genuine search traces between samples in each dataset following Valmeekam et al. (2026). Values are arithmetic means over 5 seeds, with error bars showing one standard error of the mean.

sample efficiency. Thus, the method amounts to expanding the dataset with auxiliary tasks, which can benefit sample efficiency when the main task is hard to learn, but not compute efficiency, since every additional task in the dataset grows training compute proportionately.

4.4 Scaling regime-aware data mixtures saves compute

So far, we have demonstrated the effects of scaling training data and model size on individual search tasks. In this section, we show the effects of scaling when training on a mixture of search tasks, and how the information about the scaling regime of SA models on a given task can be used in practice to minimise training compute.

We use the same dataset and model setup as in Sections 4.1 and 4.2, except that we concatenate the datasets and train 8-layer models to solve both maze navigation and modular Countdown. Since the datasets have different sizes, we repeat the maze data 4 times and use 60% of the Countdown dataset, resulting in a 24 million-sample dataset with 1:1 task balance.

We then train a model on a mixture informed by the previously identified scaling regimes: maze navigation without search traces (SO maze) and modular Countdown with search traces (SA Countdown). We compare this regime-aware mixture with two baselines: one in which both tasks use SO supervision, and another in which both tasks use SA supervision. These baselines represent a uniform choice of supervision format without knowledge of the scaling regime of each task.

Scaling regimes help derive a compute-optimal data mixture (Figure 5). Our results show that the regime-aware model outperforms both baselines on training efficiency, reaching generalisation with 1.8x less training compute than the next most efficient mixture (Figure 5). This shows that identifying the scaling regime of individual tasks can translate into compute efficiency gains when training on a heterogeneous mixture of tasks.

5 Conclusion

In this work, we studied when imitating sequential search in-context is worth the additional training compute it requires. We explored this by evaluating search-augmented (SA) models with various traces, and comparing them with solution-only (SO) models, while scaling the model size and training data.

Our results highlight a trade-off between the two supervision formats. SO models can solve problems quickly by relying on the in-context graph, local transition rules, and heuristics, but scale poorly if these are not present. SA models do not rely on these, but scale inversely to the length of the search trace, which itself depends on the size of the problem graph. Specifically, on maze navigation, SO models scale well, generalise with less training compute than SA models, and, once

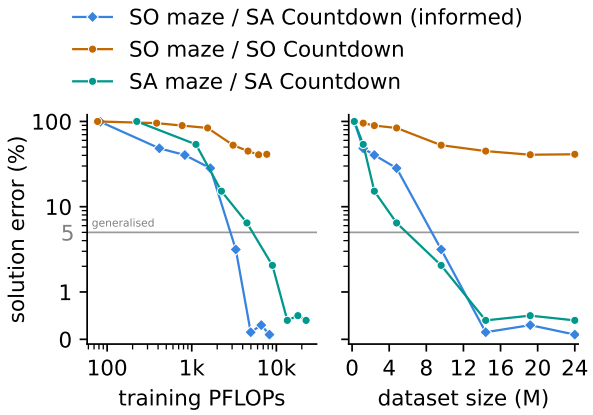


Figure 5: Regime-aware data mixing improves training efficiency across tasks. Models are trained jointly on maze navigation and modular Countdown with a 1:1 task balance. The regime-aware mixture (SO maze / SA Countdown) reaches low error with less training compute than either uniform SO or uniform SA supervision.

models are large enough, with comparable dataset sizes. On Countdown, SO models scale poorly, while SA models with optimal traces generalise with a fraction of the data and compute required by SO models; scaling model size does not change their sample efficiency. Furthermore, keeping the dataset and model sizes fixed, the performance of SA models also scales with trace quality, while the performance of SO models scales by adding useful heuristics. Specifically, we show that removing heuristics from Countdown further hurts the efficiency of SO models but does not affect SA models, and that training SA models with sub-optimal or unrelated traces hurts their performance on all the tasks and can make them less efficient than SO models.

Together, these results imply that SA supervision is not universally better or worse than SO supervision, and that awareness of the task structure and trace quality dictates when the additional costs of SA supervision are warranted. We show how a data mixture that takes the efficiency of both supervision formats into account can yield high-performing models with more restrictive compute and data budgets. Future work can further investigate some of these observations, including giving a mechanistic interpretation of how structural properties of a task affect the efficiency of SA and SO models, or which auxiliary tasks in the trace can improve sample efficiency the most.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data sets and implement methods. We have not used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, or support qualitative and thematic data analysis; helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, and cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or editing software code, sourcing/searching for information, identifying relevant literature, creating figures for the paper, and writing the Appendix. We have reviewed all AI-assisted work. We checked all the code that AI wrote, verified every experiment configuration it prepared from our instructions, and monitored all experiments manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to facilitate the reproducibility of our results. Section 3 describes the two tasks, the trace types, and the training and evaluation protocol. Appendix A gives full details of the data generation for maze navigation (Appendix A.1) and for both Countdown variants

(Appendix A.2), the search-space and trace statistics of every dataset (Appendix A.4), the serialised Countdown training records (Appendix C), and examples of generated traces (Appendices A.5 and A.6). Appendix B lists the model configurations, training hyperparameters, and hardware used for every run, and Appendix D reports the corresponding training curves. All reported metrics are computed on held-out evaluation sets of 1,000 examples per task, and every result is averaged over 5 random seeds unless otherwise stated. We will release the data generators, the training and evaluation code, and the configuration files for all experiments upon publication.

REFERENCES

Emmanuel Abbe, Samy Bengio, Aryo Lotfi, Colin Sandon, and Omid Saremi. How Far Can Transformers Reason? The Globality Barrier and Inductive Scratchpad. In *Advances in Neural Information Processing Systems*, 2024.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *International Conference on Machine Learning*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, 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. *Nature*, 645(8081):633–638, September 2025. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-025-09422-z.

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. In *Conference on Language Modeling*, 2024.

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. In *Conference on Language Modeling*, 2025.

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. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, 2022.

Liwei Kang, Yee Whye Teh, and Wee Sun Lee. LinTree: Improving LLM Reasoning with Explicitly Structured Search Histories, 2026.

Joongwon Kim, Anirudh Goyal, Liang Tan, Hannaneh Hajishirzi, Srinivasan Iyer, and Tianlu Wang. ASTRO: Teaching Language Models to Reason by Reflecting and Backtracking In-Context, July 2025.

Tughanbulut Kurtulush. When chain-of-thought helps and when it hurts: An empirical investigation of the serial-depth bottleneck in llm reasoning. *arXiv preprint arXiv:2608.09942*, 2026.

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. In *Conference on Language Modeling*, 2024.

Ryan Liu, Jiayi Geng, Addison J Wu, Ilia Sucholutsky, Tania Lombrozo, and Thomas L Griffiths. Mind your step (by step): Chain-of-thought can reduce performance on tasks where thinking makes humans worse. In *International Conference on Machine Learning*, 2025.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models. In *ICLR 2022 Workshop on Deep Learning for Code*, 2022.

Andreas Opedal, Francesco Ignazio Re, Abulhair Saparov, Mrinmaya Sachan, Bernhard Schölkopf, and Ryan Cotterell. Learning to Reason Efficiently with A* Post-Training, May 2026.

Tian Qin, David Alvarez-Melis, Samy Jelassi, and Eran Malach. To Backtrack or Not to Backtrack: When Sequential Search Limits Model Reasoning. In *Conference on Language Modeling*, 2025.

Abulhair Saparov, Srushti Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Seyed Mehran Kazemi, Najoung Kim, and He He. Transformers Struggle to Learn to Search. In *International Conference on Learning Representations*, 2025.

John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, Tom Zahavy, Petar Veličković, Laurel Prince, Satinder Singh, Eric Malmi, and Nenad Tomašev. Mastering Board Games by External and Internal Planning with Language Models. In *International Conference on Machine Learning*, 2025.

DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable Fast and Slow Thinking by Learning with Randomized Reasoning Traces. In *International Conference on Learning Representations*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He,

Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi K3: Open Frontier Intelligence, August 2026.

Karthik Valmeekam, Vardhan Palod, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Beyond Semantics: The Unreasonable Effectiveness of Reasonless Intermediate Tokens. *Transactions on Machine Learning Research*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, 2022.

Kaiyue Wen, Huaqing Zhang, Hongzhou Lin, and Jingzhao Zhang. From Sparse Dependence to Sparse Attention: Unveiling How Chain-of-Thought Enhances Transformer Sample Efficiency. In *International Conference on Learning Representations*, 2025.

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*, STOC '96, pp.

296–303, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237880. URL https://doi.org/10.1145/237814.237880.

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, Louis Castricato, Jan-Philipp Franken, Nick Haber, and Chelsea Finn. Towards System 2 Reasoning in LLMs: Learning How to Think With Meta Chain-of-Thought, January 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, May 2025a.

Chenxiao Yang, Zhiyuan Li, and David Wipf. Chain-of-Thought Provably Enables Learning the (Otherwise) Unlearnable. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=N6pbLYLeej.

Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of Thought Imitation with Procedure Cloning. In *Advances in Neural Information Processing Systems*, 2022.

Tong Yang, Yu Huang, Yingbin Liang, and Yuejie Chi. Multi-head Transformers Provably Learn Symbolic Multi-step Reasoning via Gradient Descent. In *Advances in Neural Information Processing Systems*, 2025c.

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. In *Advances in Neural Information Processing Systems*, 2023.

Qilin Ye, Deqing Fu, Robin Jia, and Vatsal Sharan. Transformers Provably Learn Algorithmic Solutions for Graph Connectivity, But Only with the Right Data. In *International Conference on Machine Learning*, 2026.

A DATASETS

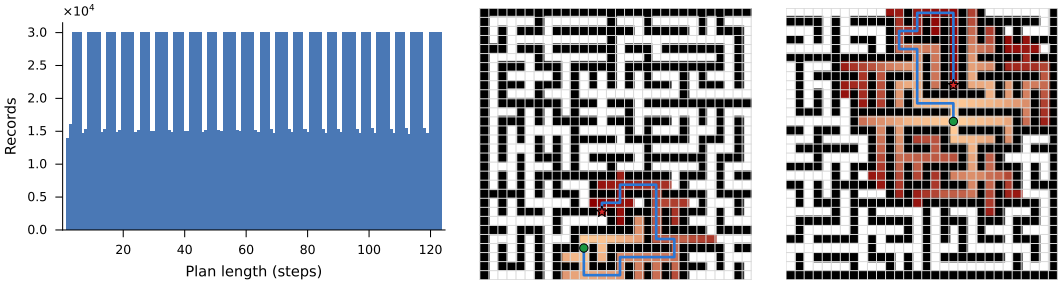
A.1 MAZE NAVIGATION

Generation. Mazes are produced with Wilson's algorithm (loop-erased random walks) on a 15×15 lattice of rooms and rendered on a 30×30 grid: rooms occupy every other cell and the 224 carved corridor cells join them, so every maze is a spanning tree with exactly 449 floor and 451 wall cells (verified on all 3,000,000 training records). On average a maze has 65 dead ends, 49 three-way and 7 four-way junctions; the remaining 328 floor cells are corridor cells with two neighbours. The reference solver is A* with the Manhattan heuristic. The corpus is *balanced by plan length*: the generator draws 10,000 calibration mazes, takes the plan-length range from the minimum to the 99th percentile (2–123 steps), splits it into 100 equal-width bins, and keeps drawing mazes (rejection sampling, seed 0) until every bin holds exactly 30,000 training records. Bins that cover two plan lengths split their 30,000 records between them, which produces the comb in Figure 6a.

Splits. 3,000,000 training records and 1,000 test records. All maze results in the paper are measured on a held-out set of 1,000 mazes produced by the same generator and settings from a different seed; none of its 1,000 wall layouts occurs among the 3,000,000 training mazes.

Vocabulary. The corpus uses 163 distinct tokens: 30 coordinates (`0`–`29`), 123 cost tokens (`c0`–`c122`) and 10 keywords (`wall`, `query`, `start`, `goal`, `reasoning`, `close`, `create`, `solution`, `plan`, `end`). All models use one word-level tokenizer of 2,619 entries shared by both domains (integers `0`–`1000`, `c0`–`c1599`, the keywords, four operators and four special tokens).

Record layout. A record is `wall` x y $\times$ 451, `query start` x_s y_s `goal` x_g y_g, `reasoning` followed by A* events of five tokens each (`close` or `create`, the cell, its path cost `c`g and heuristic `c`h), then `solution`, `plan` x y per step of the path (start and goal included) and `end`. The prompt (walls and query) is always 1,361 tokens; the trace averages 2,041 tokens (15–4,485) and the solution 189 tokens (6–369), for 3,593 tokens per record on average (Table 2).



(a) Plan-length histogram of the training split. (b) Example 1. (c) Example 2.

Figure 6: (a) Balanced sampling gives every one of the 100 plan-length bins exactly 30,000 records; 22 bins cover two lengths. (b, c) The two training records printed below: walls in black, A* `create` order from light to dark, the plan in blue, start (circle) and goal (star).

Two training records. Two training records with 25–35 plan steps, drawn in Figure 6b,c: one whose search length is the median of that window (Example 1) and the one with the longest search (Example 2). Most of every record is the fixed list of 451 walls.

Example 1. Plan of 35 steps, 137 search events, 2153 tokens in total (1353 of them the wall list).

```
wall 0 0 wall 1 0 wall 2 0 wall 3 0 wall 4 0 wall 5 0 wall 6 0 wall 7 0 wall 8 0 wall 9 0 wall
10 0 wall 11 0 wall 12 0 wall 13 0 wall 14 0 wall 15 0 wall 16 0 wall 17 0 wall 18 0 wall 19 0
wall 20 0 wall 21 0 wall 22 0 wall 23 0 wall 24 0 wall 25 0 wall 26 0 wall 27 0 wall 28 0 wall
29 0 wall 0 1 wall 6 1 wall 14 1 wall 18 1 wall 24 1 wall 26 1 wall 0 2 wall 2 2 wall 4 2 wall
5 2 wall 6 2 wall 8 2 wall 9 2 wall 10 2 wall 11 2 wall 12 2 wall 13 2 wall 14 2 wall 15 2
wall 16 2 wall 18 2 wall 19 2 wall 20 2 wall 21 2 wall 22 2 wall 24 2 wall 26 2 wall 27 2 wall
28 2 wall 0 3 wall 2 3 wall 14 3 wall 28 3 wall 0 4 wall 1 4 wall 2 4 wall 4 4 wall 5 4 wall 6
4 wall 8 4 wall 9 4 wall 10 4 wall 11 4 wall 12 4 wall 13 4 wall 14 4 wall 16 4 wall 17 4 wall
18 4 wall 20 4 wall 21 4 wall 22 4 wall 23 4 wall 24 4 wall 25 4 wall 26 4 wall 28 4 wall 0 5
wall 6 5 wall 14 5 wall 18 5 wall 26 5 wall 28 5 wall 0 6 wall 2 6 wall 3 6 wall 4 6 wall 6 6
wall 7 6 wall 8 6 wall 10 6 wall 11 6 wall 12 6 wall 13 6 wall 14 6 wall 16 6 wall 17 6 wall
18 6 wall 19 6 wall 20 6 wall 21 6 wall 22 6 wall 23 6 wall 24 6 wall 25 6 wall 26 6 wall 28 6
```

wall 0 7 wall 2 7 wall 8 7 wall 10 7 wall 12 7 wall 14 7 wall 16 7 wall 18 7 wall 22 7 wall 24 7 wall 28 7 wall 0 8 wall 2 8 wall 3 8 wall 4 8 wall 6 8 wall 7 8 wall 8 8 wall 10 8 wall 12 8 wall 14 8 wall 16 8 wall 18 8 wall 19 8 wall 20 8 wall 22 8 wall 24 8 wall 25 8 wall 26 8 wall 28 8 wall 0 9 wall 2 9 wall 6 9 wall 10 9 wall 22 9 wall 0 10 wall 2 10 wall 3 10 wall 4 10 wall 5 10 wall 6 10 wall 7 10 wall 8 10 wall 10 10 wall 11 10 wall 12 10 wall 14 10 wall 15 10 wall 16 10 wall 18 10 wall 19 10 wall 20 10 wall 21 10 wall 22 10 wall 23 10 wall 24 10 wall 25 10 wall 26 10 wall 27 10 wall 28 10 wall 29 10 wall 0 11 wall 4 11 wall 6 11 wall 8 11 wall 10 11 wall 16 11 wall 24 11 wall 26 11 wall 0 12 wall 2 12 wall 3 12 wall 4 12 wall 6 12 wall 8 12 wall 9 12 wall 10 12 wall 11 12 wall 12 12 wall 14 12 wall 15 12 wall 16 12 wall 18 12 wall 19 12 wall 20 12 wall 22 12 wall 24 12 wall 26 12 wall 28 12 wall 0 13 wall 2 13 wall 4 13 wall 10 13 wall 16 13 wall 18 13 wall 22 13 wall 26 13 wall 28 13 wall 0 14 wall 2 14 wall 4 14 wall 6 14 wall 8 14 wall 10 14 wall 11 14 wall 12 14 wall 13 14 wall 14 14 wall 16 14 wall 17 14 wall 18 14 wall 19 14 wall 20 14 wall 21 14 wall 22 14 wall 24 14 wall 25 14 wall 26 14 wall 28 14 wall 0 15 wall 2 15 wall 6 15 wall 8 15 wall 10 15 wall 18 15 wall 24 15 wall 28 15 wall 0 16 wall 2 16 wall 4 16 wall 6 16 wall 7 16 wall 8 16 wall 10 16 wall 12 16 wall 13 16 wall 14 16 wall 15 16 wall 16 16 wall 18 16 wall 19 16 wall 20 16 wall 21 16 wall 22 16 wall 23 16 wall 24 16 wall 26 16 wall 27 16 wall 28 16 wall 0 17 wall 4 17 wall 8 17 wall 10 17 wall 14 17 wall 18 17 wall 24 17 wall 26 17 wall 0 18 wall 2 18 wall 4 18 wall 6 18 wall 7 18 wall 8 18 wall 9 18 wall 10 18 wall 11 18 wall 12 18 wall 13 18 wall 14 18 wall 16 18 wall 17 18 wall 18 18 wall 19 18 wall 20 18 wall 22 18 wall 23 18 wall 24 18 wall 26 18 wall 28 18 wall 29 18 wall 0 19 wall 2 19 wall 4 19 wall 6 19 wall 8 19 wall 12 19 wall 22 19 wall 26 19 wall 0 20 wall 1 20 wall 2 20 wall 3 20 wall 4 20 wall 6 20 wall 8 20 wall 9 20 wall 10 20 wall 12 20 wall 13 20 wall 14 20 wall 16 20 wall 17 20 wall 18 20 wall 20 20 wall 21 20 wall 22 20 wall 23 20 wall 24 20 wall 26 20 wall 27 20 wall 28 20 wall 29 20 wall 0 21 wall 2 21 wall 4 21 wall 10 21 wall 18 21 wall 24 21 wall 28 21 wall 0 22 wall 2 22 wall 4 22 wall 5 22 wall 6 22 wall 8 22 wall 9 22 wall 10 22 wall 11 22 wall 12 22 wall 14 22 wall 16 22 wall 17 22 wall 18 22 wall 20 22 wall 22 22 wall 23 22 wall 24 22 wall 26 22 wall 28 22 wall 0 23 wall 8 23 wall 14 23 wall 16 23 wall 18 23 wall 20 23 wall 22 23 wall 26 23 wall 0 24 wall 1 24 wall 2 24 wall 4 24 wall 5 24 wall 6 24 wall 8 24 wall 10 24 wall 11 24 wall 12 24 wall 13 24 wall 14 24 wall 15 24 wall 16 24 wall 18 24 wall 20 24 wall 21 24 wall 22 24 wall 23 24 wall 24 24 wall 26 24 wall 27 24 wall 28 24 wall 29 24 wall 0 25 wall 4 25 wall 6 25 wall 10 25 wall 18 25 wall 0 26 wall 2 26 wall 4 26 wall 6 26 wall 8 26 wall 9 26 wall 10 26 wall 12 26 wall 14 26 wall 15 26 wall 16 26 wall 17 26 wall 18 26 wall 19 26 wall 20 26 wall 22 26 wall 23 26 wall 24 26 wall 25 26 wall 26 26 wall 28 26 wall 29 26 wall 0 27 wall 2 27 wall 4 27 wall 6 27 wall 8 27 wall 12 27 wall 14 27 wall 22 27 wall 26 27 wall 0 28 wall 2 28 wall 3 28 wall 4 28 wall 6 28 wall 8 28 wall 9 28 wall 10 28 wall 12 28 wall 13 28 wall 14 28 wall 16 28 wall 18 28 wall 20 28 wall 22 28 wall 23 28 wall 24 28 wall 26 28 wall 28 28 wall 0 29 wall 6 29 wall 8 29 wall 16 29 wall 18 29 wall 20 29 wall 22 29 wall 28 29 query start 11 26 goal 13 22 reasoning close 11 26 c0 c6 create 11 25 c1 c5 create 11 27 c1 c7 close 11 25 c1 c5 create 12 25 c2 c4 close 12 25 c2 c4 create 13 25 c3 c3 close 13 25 c3 c3 create 14 25 c4 c4 create 13 26 c4 c4 close 11 27 c1 c7 create 11 28 c2 c8 create 10 27 c2 c8 close 14 25 c4 c4 create 15 25 c5 c5 close 13 26 c4 c4 create 13 27 c5 c5 close 11 28 c2 c8 create 11 29 c3 c9 close 10 27 c2 c8 create 9 27 c3 c9 close 15 25 c5 c5 create 16 25 c6 c6 close 13 27 c5 c5 close 11 29 c3 c9 create 12 29 c4 c8 create 10 29 c4 c10 close 9 27 c3 c9 close 12 29 c4 c8 create 13 29 c5 c7 close 13 29 c5 c7 create 14 29 c6 c8 close 16 25 c6 c6 create 17 25 c7 c7 close 10 29 c4 c10 create 9 29 c5 c11 close 14 29 c6 c8 create 15 29 c7 c9 close 17 25 c7 c7 create 17 24 c8 c6 close 17 24 c8 c6 create 17 23 c9 c5 close 17 23 c9 c5 close 9 29 c5 c11 close 15 29 c7 c9 create 15 28 c8 c8 close 15 28 c8 c8 create 15 27 c9 c7 close 15 27 c9 c7 create 16 27 c10 c8 close 16 27 c10 c8 create 17 27 c11 c9 close 17 27 c11 c9 create 18 27 c12 c10 create 17 28 c12 c10 close 18 27 c12 c10 create 19 27 c13 c11 close 17 28 c12 c10 create 17 29 c13 c11 close 19 27 c13 c11 create 20 27 c14 c12 create 19 28 c14 c12 close 17 29 c13 c11 close 20 27 c14 c12 create 21 27 c15 c13 close 19 28 c14 c12 create 19 29 c15 c13 close 21 27 c15 c13 create 21 26 c16 c12 create 21 28 c16 c14 close 19 29 c15 c13 close 21 26 c16 c12 create 21 25 c17 c11 close 21 25 c17 c11 create 22 25 c18 c12 create 20 25 c18 c10 close 20 25 c18 c10 create 19 25 c19 c9 close 19 25 c19 c9 create 19 24 c20 c8 close 19 24 c20 c8 create 19 23 c21 c7 close 19 23 c21 c7 create 19 22 c22 c6 close 19 22 c22 c6 create 19 21 c23 c7 close 21 28 c16 c14 create 21 29 c17 c15 close 22 25 c18 c12 create 23 25 c19 c13 close 19 21 c23 c7 create 19 20 c24 c8 create 20 21 c24 c8 close 21 29 c17 c15 close 23 25 c19 c13 create 24 25 c20 c14 close 20 21 c24 c8 create 21 21 c25 c9 close 19 20 c24 c8 create 19 19 c25 c9 close 24 25 c20 c14 create 25 25 c21 c15 close 21 21 c25 c9 create 22 21 c26 c10 create 21 22 c26 c8 close 19 19 c25 c9 create 20 19 c26 c10 create 18 19 c26 c8 close 21 22 c26 c8 create 21 23 c27 c9 close 18 19 c26 c8 create 17 19 c27 c7 close 17 19 c27 c7 create 16 19 c28 c6 close 16 19 c28 c6 create 15 19 c29 c5 close 15 19 c29 c5 create 15 18 c30 c6 create 15 20 c30 c4 create 14 19 c30 c4 close 15 20 c30 c4 create 15 21 c31 c3 close 14 19 c30 c4 create 13 19 c31 c3 close 15 21 c31 c3 create 16 21 c32 c4 create 15 22 c32 c2 create 14 21 c32 c2 close 13 19 c31 c3 close 15 22 c32 c2 create 15 23 c33 c3 close 14 21 c32 c2 create 13 21 c33 c1 close 13 21 c33 c1 create 13 22 c34 c0 create 12 21 c34 c2 close 13 22 c34 c0 solution plan 11 26 plan 11 27 plan 11 28 plan 11 29 plan 12 29 plan 13 29 plan 14 29 plan 15 29 plan 15 28 plan 15 27 plan 16 27 plan 17 27 plan 18 27 plan 19 27 plan 20 27 plan 21 27 plan 21 26 plan 21 25 plan 20 25 plan 19 25 plan 19 24 plan 19 23 plan 19 22 plan 19 21 plan 19 20 plan 19 19 plan 18 19 plan 17 19 plan 16 19 plan 15 19 plan 15 20 plan 15 21 plan 14 21 plan 13 21 plan 13 22 end

Example 2. Plan of 33 steps, 390 search events, 3412 tokens in total (1353 of them the wall list).

wall 13 0 wall 19 0 wall 21 0 wall 29 0 wall 0 1 wall 1 1 wall 2 1 wall 3 1 wall 5 1 wall 7 1 wall 8 1 wall 9 1 wall 10 1 wall 11 1 wall 13 1 wall 15 1 wall 17 1 wall 19 1 wall 21 1 wall 22 1 wall 23 1 wall 24 1 wall 25 1 wall 26 1 wall 27 1 wall 29 1 wall 3 2 wall 5 2 wall 11 2 wall 15 2 wall 17 2 wall 19 2 wall 21 2 wall 23 2 wall 25 2 wall 29 2 wall 0 3 wall 1 3 wall 3 3 wall 5 3 wall 7 3 wall 8 3 wall 9 3 wall 11 3 wall 13 3 wall 14 3 wall 15 3 wall 16 3 wall 17 3 wall 19 3 wall 21 3 wall 23 3 wall 25 3 wall 27 3 wall 28 3 wall 29 3 wall 5 4 wall 9 4

wall 11 4 wall 17 4 wall 19 4 wall 23 4 wall 29 4 wall 0 5 wall 1 5 wall 2 5 wall 3 5 wall 4 5
wall 5 5 wall 6 5 wall 7 5 wall 8 5 wall 9 5 wall 10 5 wall 11 5 wall 12 5 wall 13 5 wall 15 5
wall 17 5 wall 19 5 wall 21 5 wall 22 5 wall 23 5 wall 24 5 wall 25 5 wall 27 5 wall 28 5 wall
29 5 wall 1 6 wall 5 6 wall 15 6 wall 17 6 wall 19 6 wall 23 6 wall 29 6 wall 1 7 wall 3 7
wall 4 7 wall 5 7 wall 6 7 wall 7 7 wall 9 7 wall 11 7 wall 12 7 wall 13 7 wall 15 7 wall 17 7
wall 19 7 wall 21 7 wall 22 7 wall 23 7 wall 24 7 wall 25 7 wall 26 7 wall 27 7 wall 29 7 wall
1 8 wall 5 8 wall 9 8 wall 11 8 wall 15 8 wall 17 8 wall 19 8 wall 23 8 wall 25 8 wall 29 8
wall 1 9 wall 3 9 wall 5 9 wall 7 9 wall 9 9 wall 11 9 wall 13 9 wall 15 9 wall 16 9 wall 17 9
wall 18 9 wall 19 9 wall 20 9 wall 21 9 wall 23 9 wall 25 9 wall 27 9 wall 29 9 wall 1 10 wall
3 10 wall 5 10 wall 7 10 wall 9 10 wall 11 10 wall 13 10 wall 27 10 wall 29 10 wall 1 11 wall
2 11 wall 3 11 wall 5 11 wall 7 11 wall 8 11 wall 9 11 wall 10 11 wall 11 11 wall 12 11 wall
13 11 wall 14 11 wall 15 11 wall 16 11 wall 17 11 wall 19 11 wall 20 11 wall 21 11 wall 22 11
wall 23 11 wall 24 11 wall 25 11 wall 27 11 wall 29 11 wall 7 12 wall 25 12 wall 27 12 wall 29
12 wall 0 13 wall 1 13 wall 3 13 wall 4 13 wall 5 13 wall 7 13 wall 8 13 wall 9 13 wall 11 13
wall 13 13 wall 14 13 wall 15 13 wall 16 13 wall 17 13 wall 19 13 wall 20 13 wall 21 13 wall
22 13 wall 23 13 wall 25 13 wall 26 13 wall 27 13 wall 28 13 wall 29 13 wall 3 14 wall 9 14
wall 11 14 wall 13 14 wall 17 14 wall 23 14 wall 25 14 wall 29 14 wall 1 15 wall 2 15 wall 3
15 wall 4 15 wall 5 15 wall 6 15 wall 7 15 wall 9 15 wall 11 15 wall 13 15 wall 14 15 wall 15
15 wall 17 15 wall 18 15 wall 19 15 wall 21 15 wall 22 15 wall 23 15 wall 24 15 wall 25 15
wall 27 15 wall 28 15 wall 29 15 wall 3 16 wall 9 16 wall 11 16 wall 19 16 wall 21 16 wall 23
16 wall 29 16 wall 0 17 wall 1 17 wall 2 17 wall 3 17 wall 5 17 wall 7 17 wall 8 17 wall 9 17
wall 11 17 wall 13 17 wall 15 17 wall 16 17 wall 17 17 wall 18 17 wall 19 17 wall 21 17 wall
23 17 wall 24 17 wall 25 17 wall 27 17 wall 29 17 wall 5 18 wall 7 18 wall 11 18 wall 13 18
wall 15 18 wall 17 18 wall 25 18 wall 27 18 wall 29 18 wall 0 19 wall 1 19 wall 3 19 wall 5 19
wall 6 19 wall 7 19 wall 8 19 wall 9 19 wall 11 19 wall 13 19 wall 14 19 wall 15 19 wall 17 19
wall 18 19 wall 19 19 wall 21 19 wall 23 19 wall 25 19 wall 27 19 wall 28 19 wall 29 19 wall 1
20 wall 3 20 wall 5 20 wall 7 20 wall 11 20 wall 13 20 wall 21 20 wall 23 20 wall 29 20 wall 1
21 wall 3 21 wall 4 21 wall 5 21 wall 7 21 wall 8 21 wall 9 21 wall 10 21 wall 11 21 wall 12
21 wall 13 21 wall 15 21 wall 16 21 wall 17 21 wall 18 21 wall 19 21 wall 20 21 wall 21 21
wall 23 21 wall 24 21 wall 25 21 wall 26 21 wall 27 21 wall 29 21 wall 5 22 wall 7 22 wall 9
22 wall 15 22 wall 17 22 wall 21 22 wall 23 22 wall 27 22 wall 29 22 wall 1 23 wall 2 23 wall
3 23 wall 4 23 wall 5 23 wall 7 23 wall 9 23 wall 10 23 wall 11 23 wall 12 23 wall 13 23 wall
15 23 wall 17 23 wall 18 23 wall 19 23 wall 21 23 wall 22 23 wall 23 23 wall 25 23 wall 26 23
wall 27 23 wall 29 23 wall 1 24 wall 7 24 wall 13 24 wall 15 24 wall 17 24 wall 19 24 wall 21
24 wall 25 24 wall 29 24 wall 0 25 wall 1 25 wall 3 25 wall 4 25 wall 5 25 wall 6 25 wall 7 25
wall 9 25 wall 11 25 wall 12 25 wall 13 25 wall 15 25 wall 17 25 wall 19 25 wall 21 25 wall 22
25 wall 23 25 wall 25 25 wall 26 25 wall 27 25 wall 29 25 wall 9 26 wall 15 26 wall 19 26 wall
27 26 wall 29 26 wall 1 27 wall 3 27 wall 4 27 wall 5 27 wall 6 27 wall 7 27 wall 9 27 wall 10
27 wall 11 27 wall 13 27 wall 15 27 wall 17 27 wall 18 27 wall 19 27 wall 20 27 wall 21 27
wall 22 27 wall 23 27 wall 25 27 wall 27 27 wall 29 27 wall 1 28 wall 7 28 wall 11 28 wall 13
28 wall 15 28 wall 25 28 wall 29 28 wall 0 29 wall 1 29 wall 2 29 wall 3 29 wall 4 29 wall 5
29 wall 6 29 wall 7 29 wall 8 29 wall 9 29 wall 10 29 wall 11 29 wall 12 29 wall 13 29 wall 14
29 wall 15 29 wall 16 29 wall 17 29 wall 18 29 wall 19 29 wall 20 29 wall 21 29 wall 22 29
wall 23 29 wall 24 29 wall 25 29 wall 26 29 wall 27 29 wall 28 29 wall 29 29 query start 18 12
goal 18 8 reasoning close 18 12 c0 c4 create 18 11 c1 c3 create 19 12 c1 c5 create 18 13 c1 c5
create 17 12 c1 c5 close 18 11 c1 c3 create 18 10 c2 c2 close 18 10 c2 c2 create 19 10 c3 c3
create 17 10 c3 c3 close 17 12 c1 c5 create 16 12 c2 c6 close 19 12 c1 c5 create 20 12 c2 c6
close 18 13 c1 c5 create 18 14 c2 c6 close 17 10 c3 c3 create 16 10 c4 c4 close 19 10 c3 c3
create 20 10 c4 c4 close 20 12 c2 c6 create 21 12 c3 c7 close 16 12 c2 c6 create 15 12 c3 c7
close 18 14 c2 c6 create 19 14 c3 c7 close 16 10 c4 c4 create 15 10 c5 c5 close 20 10 c4 c4
create 21 10 c5 c5 close 15 12 c3 c7 create 14 12 c4 c8 close 21 12 c3 c7 create 22 12 c4 c8
close 19 14 c3 c7 create 20 14 c4 c8 close 15 10 c5 c5 create 14 10 c6 c6 close 21 10 c5 c5
create 22 10 c6 c6 close 22 12 c4 c8 create 23 12 c5 c9 close 14 12 c4 c8 create 13 12 c5 c9
close 20 14 c4 c8 create 21 14 c5 c9 create 20 15 c5 c9 close 14 10 c6 c6 create 14 9 c7 c5
close 22 10 c6 c6 create 22 9 c7 c5 create 23 10 c7 c7 close 14 9 c7 c5 create 14 8 c8 c4
close 22 9 c7 c5 create 22 8 c8 c4 close 14 8 c8 c4 create 14 7 c9 c5 create 13 8 c9 c5 close
22 8 c8 c4 create 21 8 c9 c3 close 21 8 c9 c3 create 20 8 c10 c2 close 20 8 c10 c2 create 20 7
c11 c3 close 13 12 c5 c9 create 12 12 c6 c10 close 23 12 c5 c9 create 24 12 c6 c10 close 21 14
c5 c9 create 22 14 c6 c10 close 20 15 c5 c9 create 20 16 c6 c10 close 23 10 c7 c7 create 24 10
c8 c8 close 14 7 c9 c5 create 14 6 c10 c6 close 13 8 c9 c5 create 12 8 c10 c6 close 20 7 c11
c3 create 20 6 c12 c4 close 22 14 c6 c10 close 20 16 c6 c10 create 20 17 c7 c11 close 24 12 c6
c10 create 24 13 c7 c11 close 12 12 c6 c10 create 12 13 c7 c11 create 11 12 c7 c11 close 24 10
c8 c8 create 24 9 c9 c7 create 25 10 c9 c9 close 24 9 c9 c7 create 24 8 c10 c6 close 12 8 c10
c6 create 12 9 c11 c7 close 14 6 c10 c6 create 14 5 c11 c7 create 13 6 c11 c7 close 24 8 c10
c6 close 20 6 c12 c4 create 20 5 c13 c5 create 21 6 c13 c5 close 12 13 c7 c11 create 12 14 c8
c12 close 11 12 c7 c11 create 10 12 c8 c12 close 24 13 c7 c11 create 24 14 c8 c12 close 20 17
c7 c11 create 20 18 c8 c12 close 25 10 c9 c9 create 26 10 c10 c10 close 14 5 c11 c7 create 14
4 c12 c8 close 13 6 c11 c7 create 12 6 c12 c8 close 12 9 c11 c7 create 12 10 c12 c8 close 21 6
c13 c5 create 22 6 c14 c6 close 20 5 c13 c5 create 20 4 c14 c6 close 12 14 c8 c12 create 12 15
c9 c13 close 20 18 c8 c12 create 21 18 c9 c13 create 20 19 c9 c13 create 19 18 c9 c11 close 24
14 c8 c12 close 10 12 c8 c12 create 10 13 c9 c13 create 9 12 c9 c13 close 19 18 c9 c11 create
18 18 c10 c10 close 26 10 c10 c10 create 26 9 c11 c9 create 26 11 c11 c11 close 18 18 c10 c10
close 26 9 c11 c9 create 26 8 c12 c8 close 12 6 c12 c8 create 11 6 c13 c9 close 26 8 c12 c8
create 27 8 c13 c9 close 12 10 c12 c8 close 14 4 c12 c8 create 15 4 c13 c7 create 13 4 c13 c9
close 15 4 c13 c7 create 16 4 c14 c6 close 20 4 c14 c6 create 20 3 c15 c7 create 21 4 c15 c7
close 16 4 c14 c6 create 16 5 c15 c5 close 22 6 c14 c6 close 16 5 c15 c5 create 16 6 c16 c4
close 16 6 c16 c4 create 16 7 c17 c3 close 16 7 c17 c3 create 16 8 c18 c2 close 16 8 c18 c2
close 20 19 c9 c13 create 20 20 c10 c14 close 9 12 c9 c13 create 8 12 c10 c14 close 10 13 c9
c13 create 10 14 c10 c14 close 21 18 c9 c13 create 22 18 c10 c14 close 12 15 c9 c13 create 12
16 c10 c14 close 26 11 c11 c11 create 26 12 c12 c12 close 13 4 c13 c9 create 12 4 c14 c10
close 27 8 c13 c9 create 28 8 c14 c10 close 11 6 c13 c9 create 10 6 c14 c10 close 20 3 c15 c7

```
create 20 2 c16 c8 close 21 4 c15 c7 create 22 4 c16 c8 close 22 18 c10 c14 create 22 17 c11
c13 create 23 18 c11 c15 create 22 19 c11 c15 close 12 16 c10 c14 create 13 16 c11 c13 create
12 17 c11 c15 close 20 20 c10 c14 create 19 20 c11 c13 close 8 12 c10 c14 close 10 14 c10 c14
create 10 15 c11 c15 close 13 16 c11 c13 create 14 16 c12 c12 close 19 20 c11 c13 create 18 20
c12 c12 close 22 17 c11 c13 create 22 16 c12 c12 close 14 16 c12 c12 create 15 16 c13 c11
create 14 17 c13 c13 close 18 20 c12 c12 create 17 20 c13 c13 close 22 16 c12 c12 close 26 12
c12 c12 close 15 16 c13 c11 create 16 16 c14 c10 close 28 8 c14 c10 create 28 7 c15 c11 create
28 9 c15 c11 close 16 16 c14 c10 create 16 15 c15 c9 create 17 16 c15 c9 close 12 4 c14 c10
create 12 3 c15 c11 close 10 6 c14 c10 create 10 7 c15 c9 create 9 6 c15 c11 close 16 15 c15
c9 create 16 14 c16 c8 close 17 16 c15 c9 create 18 16 c16 c8 close 10 7 c15 c9 create 10 8
c16 c8 close 16 14 c16 c8 create 15 14 c17 c9 close 18 16 c16 c8 close 10 8 c16 c8 create 10 9
c17 c9 close 20 2 c16 c8 create 20 1 c17 c9 close 22 4 c16 c8 create 22 3 c17 c9 close 10 15
c11 c15 create 10 16 c12 c16 close 12 17 c11 c15 create 12 18 c12 c16 close 23 18 c11 c15
create 24 18 c12 c16 close 22 19 c11 c15 create 22 20 c12 c16 close 17 20 c13 c13 create 16 20
c14 c14 close 14 17 c13 c13 create 14 18 c14 c14 close 9 6 c15 c11 create 8 6 c16 c12 close 28
7 c15 c11 create 28 6 c16 c12 close 12 3 c15 c11 create 12 2 c16 c12 close 28 9 c15 c11 create
28 10 c16 c12 close 20 1 c17 c9 create 20 0 c18 c10 close 15 14 c17 c9 create 14 14 c18 c10
close 10 9 c17 c9 create 10 10 c18 c10 close 22 3 c17 c9 create 22 2 c18 c10 close 24 18 c12
c16 create 24 19 c13 c17 close 22 20 c12 c16 create 22 21 c13 c17 close 12 18 c12 c16 create
12 19 c13 c17 close 10 16 c12 c16 create 10 17 c13 c17 close 14 18 c14 c14 close 16 20 c14 c14
create 16 19 c15 c13 create 15 20 c15 c15 close 16 19 c15 c13 create 16 18 c16 c12 close 8 6
c16 c12 create 8 7 c17 c11 create 7 6 c17 c13 close 12 2 c16 c12 create 12 1 c17 c13 create 13
2 c17 c11 close 16 18 c16 c12 close 28 10 c16 c12 create 28 11 c17 c13 close 28 6 c16 c12
create 27 6 c17 c11 close 8 7 c17 c11 create 8 8 c18 c10 close 27 6 c17 c11 create 26 6 c18
c10 close 13 2 c17 c11 create 14 2 c18 c10 close 8 8 c18 c10 create 8 9 c19 c11 create 7 8 c19
c11 close 26 6 c18 c10 create 26 5 c19 c11 create 25 6 c19 c9 close 14 2 c18 c10 create 14 1
c19 c11 close 14 14 c18 c10 close 22 2 c18 c10 close 10 10 c18 c10 close 20 0 c18 c10 close 25
6 c19 c9 create 24 6 c20 c8 close 24 6 c20 c8 close 10 17 c13 c17 create 10 18 c14 c18 close
12 19 c13 c17 create 12 20 c14 c18 close 24 19 c13 c17 create 24 20 c14 c18 close 22 21 c13
c17 create 22 22 c14 c18 close 15 20 c15 c15 create 14 20 c16 c16 close 28 11 c17 c13 create
28 12 c18 c14 close 7 6 c17 c13 create 6 6 c18 c14 close 12 1 c17 c13 create 12 0 c18 c14
close 14 1 c19 c11 create 14 0 c20 c12 close 7 8 c19 c11 create 6 8 c20 c12 close 8 9 c19 c11
create 8 10 c20 c12 close 26 5 c19 c11 create 26 4 c20 c12 close 24 20 c14 c18 create 25 20
c15 c19 close 22 22 c14 c18 close 12 20 c14 c18 close 10 18 c14 c18 create 10 19 c15 c19
create 9 18 c15 c19 close 14 20 c16 c16 create 14 21 c17 c17 close 6 6 c18 c14 close 28 12 c18
c14 close 12 0 c18 c14 create 11 0 c19 c15 close 26 4 c20 c12 create 26 3 c21 c13 create 27 4
c21 c13 create 25 4 c21 c11 close 6 8 c20 c12 create 6 9 c21 c13 close 8 10 c20 c12 close 14 0
c20 c12 create 15 0 c21 c11 close 25 4 c21 c11 create 24 4 c22 c10 close 15 0 c21 c11 create
16 0 c22 c10 close 24 4 c22 c10 create 24 3 c23 c11 close 16 0 c22 c10 create 17 0 c23 c9
create 16 1 c23 c9 close 17 0 c23 c9 create 18 0 c24 c8 close 16 1 c23 c9 create 16 2 c24 c8
close 18 0 c24 c8 create 18 1 c25 c7 close 16 2 c24 c8 close 18 1 c25 c7 create 18 2 c26 c6
close 18 2 c26 c6 create 18 3 c27 c5 close 18 3 c27 c5 create 18 4 c28 c4 close 18 4 c28 c4
create 18 5 c29 c3 close 18 5 c29 c3 create 18 6 c30 c2 close 18 6 c30 c2 create 18 7 c31 c1
close 18 7 c31 c1 create 18 8 c32 c0 close 18 8 c32 c0 solution plan 18 12 plan 18 11 plan 18
10 plan 17 10 plan 16 10 plan 15 10 plan 14 10 plan 14 9 plan 14 8 plan 14 7 plan 14 6 plan 14
5 plan 14 4 plan 13 4 plan 12 4 plan 12 3 plan 12 2 plan 13 2 plan 14 2 plan 14 1 plan 14 0
plan 15 0 plan 16 0 plan 17 0 plan 18 0 plan 18 1 plan 18 2 plan 18 3 plan 18 4 plan 18 5 plan
18 6 plan 18 7 plan 18 8 end
```


A.2 COUNTDOWN

We use two variants of the game, summarised in Table 1. The *regular* variant is the game of Gandhi et al. (2024); the *modular* variant is ours.

Table 1: The two Countdown variants and their corpora. "Goal reached" is the fraction of records whose stored search reaches the target; the other records end with an empty solution. Distinct tokens are counted over the training split.

	regular (Gandhi et al., 2024)	modular (ours)
starting numbers	4, from 1–99, in random order	3, from 0–1000, sorted ascending
target	10–100 (82 values)	0–1000 (all 1,001 residues)
operators	$+, -, \times, \div$	$+, -, \times$ modulo 1001
subtraction / division	$\max - \min$ only; $\div$ only when exact	both orders; no division
intermediate values	capped at 999	residues modulo 1001
target validity	reachable	reachable with all three numbers, not with any pair
operations per solution	3	2
trace corpora	SoS: 20,000,000 records DFS: 20,000,000 records (15,971,887 distinct)	DFS: 20,000,000 records (all distinct) SoS: the same 20,000,000 puzzles, re-traced
held out	1,000 per corpus	1,000 (shared by both corpora)
goal reached	SoS 59.8%; DFS 100%	DFS 100%; SoS 54.6%
distinct tokens	808 (SoS), 1,011 (DFS)	1,011

Regular Countdown. Puzzles and stream-of-search (SoS) traces are generated with the authors' code, with two changes: intermediate values are capped at 999 (their value range), and any puzzle whose SoS record exceeds 3,000 tokens in our tokenizer is redrawn. An SoS trace is produced by one of 12 search strategies drawn per puzzle, two heuristics (h_{sum}, h_{mult}) combined with a threshold-pruned depth-first search (probability $1/2$) or a beam search of width 1–5; these searches are incomplete, and only 59.8% of the training records reach the target. The SoS and the DFS corpus each hold 20,000,000 records (40 shards of 500,000 puzzles with independent seeds and the same target hold-out) and are separate draws from this generator; the DFS traces are complete depth-first searches over the same move set that never expand a pool twice. The SoS statistics reported here (goal rate, token counts, vocabulary) are measured on the corpus's first shard of 500,000 records, which the build reproduces exactly. Each corpus has its own 1,000-puzzle test split. The DFS test split is disjoint from all 20,000,000 training rows by puzzle (target and number multiset); the SoS test split follows the authors' train/validation split, in which 9 of the 1,000 test puzzles also occur in the first 500,000 training records.

Modular Countdown. Puzzles are generated by our own generator (seed 1209): three numbers are drawn from 0–1000 and shown sorted, all arithmetic is modulo 1001, and a target is kept only if it is reachable from the three numbers but not from any single number or pair, so every solution uses both operations. The 20,000,000 training puzzles are distinct (they span 812,309 distinct number triples); the 1,000 test puzzles share no number triple with the training split. The DFS corpus stores a complete depth-first search per puzzle. The SoS corpus holds the same 20,000,000 puzzles in the same order, with the reasoning region replaced by one of the 12 SoS strategies above run with the modular move set (the authors' search code, seed 0); 54.6% of these searches reach the target.

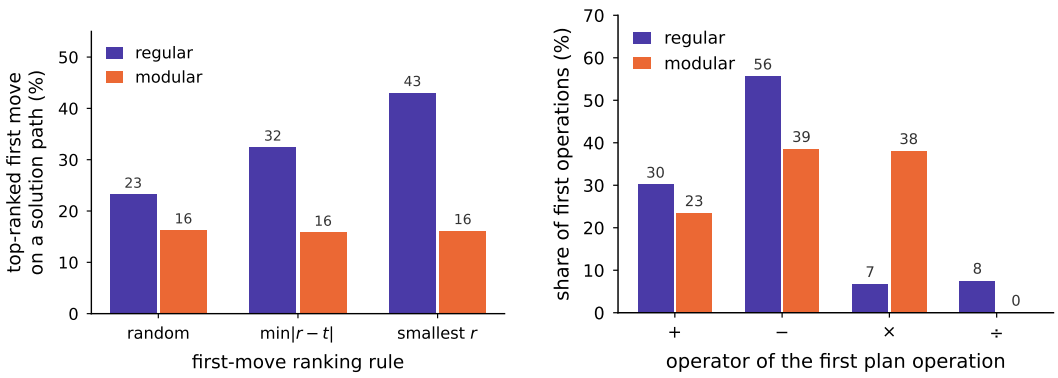
Training records. Records are $n_1 \ldots n_k$ `query` t `reasoning` followed by `close` events listing the current pool and `create` a `op` b c events for each move, then `solution`, `plan` a `op` b c per operation and `end`. The shortest records of each corpus are printed in Appendix C.

A.3 Regular Countdown vs modular Countdown

First-move heuristics. We rank the legal first operations of a puzzle by the magnitude of their result r and ask how often the top-ranked operation lies on some complete solution path (all solution paths are enumerated with the corpus's move generator; 20,000 training puzzles per variant). On regular Countdown, preferring the smallest result succeeds on 43% of puzzles and preferring the result closest to the target on 32%, against 23% for a random legal first move; on modular Countdown all three rules sit at the 16% chance rate (Figure 7).

First operation. The operator of the first reference-plan operation over the whole training split: $+$ or $-$ on 86% of regular puzzles ($\times$ on 7%, $\div$ on 8%), against 62% on modular puzzles, where $\times$ rises to 38% and subtraction, which exists in both operand orders, keeps 39%.

Second operation overlap. We key the final plan operation of each puzzle by its operand pair and result and count the distinct test keys that also occur in the training split. For regular Countdown the training split contains only 18,010 distinct final operations (35,535 in the 20M corpus): 91% of the test split's final operations already occur among the first 50,000 training records, 96% among 100,000 and 99.4% in the full 500,000. For modular Countdown the training split contains 1,987,340 distinct final operations, and the test split's are covered at 2% after 50,000 records, 29% after 550,000, 67% after 2M, 91% after 5M and 100% only with all 20M.



(a) First-move heuristics.

(b) Operator of the first plan operation.

Figure 7: Regular versus modular Countdown. (a) Share of puzzles whose top-ranked first operation, under each ranking rule, lies on a solution path. (b) Operator frequencies of the first reference-plan operation over the training split.

A.4 Search-space and trace statistics

Table 2: Search-space structure and trace statistics of every corpus. *Search space*: the tree of all move sequences from the initial state (for a maze, the maze rooted at the start cell); *max width* is the mean over puzzles of the largest number of nodes at one depth, *depth* the mean number of moves from the root to the goal, and *branching* the mean number of children per non-terminal node. Maze values are measured on 50,000 training mazes, Countdown values on each corpus's 1,000 test puzzles. *Trace*: means over all training records (†over the first 500,000-record shard for the regular SoS corpus), with the observed range in parentheses; *segments* are `close` and `create` events, *solution* and *trace tokens* count the tokens after `solution` and after `reasoning`, respectively. SoS rows include the records whose search failed (empty solution); restricted to records that reach the goal, the SoS traces average 102 segments and 384 tokens on regular and 42 segments and 152 tokens on modular Countdown.

		search space			trace				
task	corpus	max width	depth	branching	type	plan steps	segments	sol. tokens	trace tokens
maze 30 × 30	3M	10.7	62.0	1.17	A⋆	63.0 (2–123)	408 (3–897)	189	2,041
regular Countdown, 4	20M	276	3	3.08	DFS	3	107 (6–674)	15	412
regular Countdown, 4	20M	280	3	3.09	SoS†	1.8 (0–3)	94 (1–828)	9.0	344
modular Countdown, 3	20M	48	2	4.62	DFS	2	36.4 (4–76)	10	134
modular Countdown, 3	20M	48	2	4.62	SoS	1.1 (0–2)	35.6 (1–164)	5.5	124

A.5 Generated traces and plans on maze navigation

Figures 8 and 9 show the same four held-out mazes solved by the 8-layer model trained on the full corpus with A⋆ traces (`normal`) and by its counterpart trained with each record's trace replaced by the trace of a different, randomly drawn maze (`swapped`). Both models are decoded greedily.

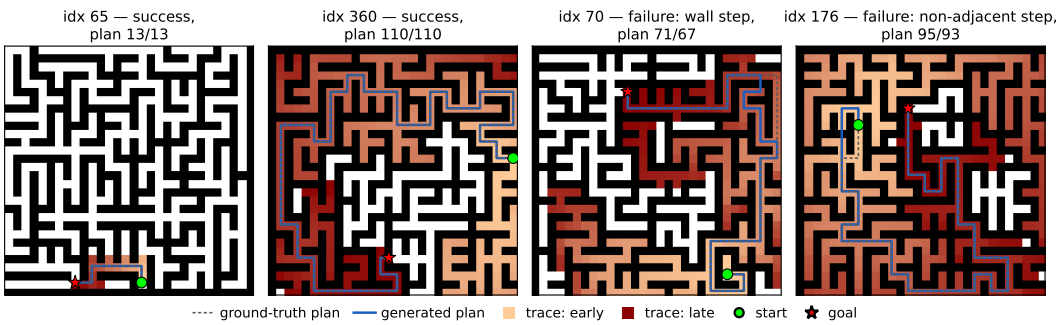


Figure 8: The 8-layer model trained with A* traces on two solved and two failed held-out mazes. Walls are black, the generated trace's `create` events are coloured from light (early) to dark (late), the generated plan is the solid blue line and the reference plan the dashed grey line; panel titles give the record index, the outcome and the generated versus reference plan length. The model solves 994 of the 1,000 held-out mazes and emits a structurally valid trace on 999; all six failures are plans that step through a wall (five) or between non-adjacent cells (one), after a trace that is valid and reaches the goal.

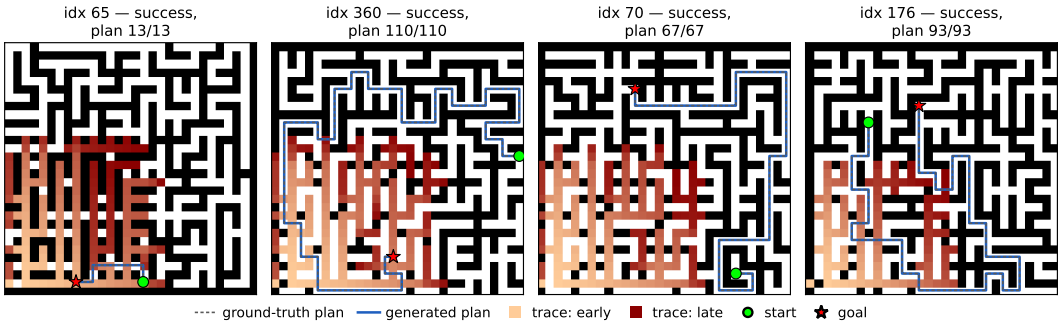


Figure 9: The `swapped` model on the same four mazes, same rendering. Its trace ignores the maze: all 1,000 generated traces start at cell $(1, 28)$ regardless of the query, only 110 distinct traces occur, and 51.7% of their `create` cells fall on walls (the wall density is 50.1%), so no trace is structurally valid. The plan that follows is nevertheless correct on 989 of 1,000 mazes, including the two mazes that the A*-trace model fails.

A.6 Generated traces and plans on Countdown

The 8-layer model trained on the full modular corpus with DFS traces, decoded greedily on the held-out puzzles. Generations are printed as emitted, colour-coded as in Appendix C; the paragraph gives the success rate and the kinds of failure observed.

Modular Countdown-3, DFS traces. 59M parameters (8 layers), seed 0, dataset fraction 1.0, one epoch; greedy decoding on the 1000-row test split with a budget of 2000 tokens and 3× the longest reference continuation of the batch. Plan success 996/1000 (99.6%); valid trace grammar 99.7%, trace structure 99.2%, plan grammar 99.8%, plan structure 99.6%. Failures by kind (4): 2 invalid plan grammar, 2 invalid plan structure; 3 of them with a trace that fails the DFS replay. (Seed 0 trained on the same 20M pool under its earlier name `countdown_3_3_max_1000_20000000_seed_1209`; scored on the `countdown_3_pmt_sorted_20m_1209` test split shared with the SoS arm.)

Example 1 (success, short trace): test row 48, 4 trace events, 28 generated tokens, reference plan 527 + 738 = 264, 264 + 770 = 33.

```
527 738 770 query 33 reasoning close 527 738 770 create 527 + 738 264 close 264 770 create 264
+ 770 33 solution plan 527 + 738 264 plan 264 + 770 33 end
```

Example 2 (success, long trace): test row 95, 86 trace events, 327 generated tokens, reference plan 557 - 290 = 267, 58 ∗ 267 = 471.

```
58 290 557 query 471 reasoning close 58 290 557 create 58 + 290 348 close 348 557 create 348 +
557 905 close 905 create 348 * 557 643 close 643 create 348 - 557 792 close 792 create 557 -
348 209 close 209 create 58 * 290 804 close 557 804 create 557 + 804 360 close 360 create 557
* 804 381 close 381 create 557 - 804 754 close 754 create 804 - 557 247 close 247 create 58 -
290 769 close 557 769 create 557 + 769 325 close 325 create 557 * 769 906 close 906 create 557
- 769 789 close 789 create 769 - 557 212 close 212 create 290 - 58 232 close 232 557 create
232 * 557 95 close 95 create 232 - 557 676 close 676 create 290 / 58 5 close 5 557 create 5 +
557 562 close 562 create 5 * 557 783 close 783 create 5 - 557 449 close 449 create 557 - 5 552
close 552 create 58 + 557 615 close 290 615 create 290 * 615 172 close 172 create 58 * 557 274
close 274 290 create 274 + 290 564 close 564 create 274 - 290 985 close 985 create 290 - 274
16 close 16 create 58 - 557 502 close 290 502 create 290 * 502 435 close 435 create 557 - 58
499 close 290 499 create 290 * 499 566 close 566 create 290 + 557 847 close 58 847 create 58 *
847 77 close 77 create 290 * 557 369 close 58 369 create 58 + 369 427 close 427 create 58 -
369 690 close 690 create 369 - 58 311 close 311 create 290 - 557 734 close 58 734 create 58 *
734 530 close 530 create 557 - 290 267 close 58 267 create 58 * 267 471 solution plan 557 -
290 267 plan 58 * 267 471 end
```

Example 3 (failure: invalid plan grammar): test row 896, 134 trace events, 498 generated tokens, reference plan 123 ∗ 596 = 235, 235 - 240 = 996 — the plan ‘plan 596 - 240’ is not grammatical; the trace is not grammatical. (trace shortened: first 60 and last 30 of 134 events shown)

```
123 240 596 query 996 reasoning close 123 240 596 create 123 + 240 363 close 363 596 create
363 + 596 959 close 959 create 363 * 596 132 close 132 create 363 - 596 768 close 768 create
596 - 363 233 close 233 create 123 * 240 491 close 491 596 create 491 + 596 86 close 86 create
491 * 596 344 close 344 create 491 - 596 896 close 896 create 596 - 491 105 close 105 create
123 - 240 884 close 596 884 create 596 + 884 479 close 479 create 596 * 884 338 close 338
create 596 - 884 713 close 713 create 884 - 596 288 close 288 create 240 - 123 117 close 117
596 create 117 * 596 663 close 663 create 117 - 596 522 close 522 create 123 + 596 719 close
240 719 create 240 * 719 388 close 388 create 123 * 596 235 close 224 240 create 224 + 240 464
close 464 create 224 - 240 985 close 985 create 240 - 224 16 close 16 create 123 - 596 528
close 240 528 create 240 * 528 594 close 594 create 596 - 123 473 close 240 473 create 240 *
473 407 close 407 create 240 + 596 836 close 123 836 create 123 * 836 726 [... 44 events omitted ...]
close 122 create 240 - 240 0 close 0 123 create 0 * 123 0 close 596 - 240 close 240 240 create
240 * 240 close 240 close 240 240 create 240 + 596 913 close 913 create 240 - 240 988 close
988 create 240 - 240 988 close 988 create 240 - 240 0 close 0 create 240 / 240 1 close 1
create 240 - 240 13 close 13 123 create 13 * 123 598 close 598 create 240 * 592 792 close 240
792 create 240 + 792 759 close 759 create 792 / 240 1 close 1 create 240 - 792 / 240 996
solution plan 596 - 240 end
```

Example 4 (failure: invalid plan structure): test row 831, 26 trace events, 107 generated tokens, reference plan 787 - 716 = 71, 71 - 788 = 284 — 787 - 787 is 0, not 71.

```
716 787 788 query 284 reasoning close 716 787 788 create 716 + 787 502 close 502 788 create
502 + 788 289 close 289 create 502 * 788 181 close 181 create 502 - 788 715 close 715 create
788 - 502 286 close 286 create 716 * 787 930 close 788 930 create 788 + 930 717 close 717
create 788 * 930 108 close 108 create 788 - 930 859 close 859 create 930 - 788 142 close 142
create 787 - 716 71 close 71 788 create 71 * 788 893 close 893 create 71 - 788 284 solution
plan 787 - 787 71 plan 71 - 788 284 end
```

B TRAINING

B.1 HYPERPARAMETERS

Table 3 lists the hyperparameters shared by every run, Table 4 the model configurations and Table 5 the runs behind each arm of the final figures.

Table 3: Hyperparameters shared by every training run behind the final figures.

Architecture	Llama-style decoder: SwiGLU feed-forward, RoPE ($\theta = 10^4$), RMSNorm ($\epsilon = 10^{-5}$), tied embeddings, head dimension 64, no dropout, initialisation std 0.02
Tokenizer / context	2,619 tokens (shared by both domains) / 32,768 tokens, no truncation
Optimizer	AdamW (fused), $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$
Weight decay / gradient clipping	0.1 / global norm 1
Peak learning rate	2×10^{-4} (every model size)
Schedule	linear warm-up over the first 1% of steps from 2×10^{-5}, then cosine decay over the remaining 99% to 2×10^{-5}; no constant phase
Precision	bf16 autocast over fp32 master weights; `torch.compile`
Seeds	0, 10, 20, 30, 40 per configuration; a seed re-draws the initial weights, the batch order and the data subset
Horizon	4 epochs of the selected data on maze, 1 epoch on Countdown (5 and 10 epochs of the full pool for the solution-only arms)
In-training evaluation	100 generations every `eval_steps`; all reported metrics come from the offline evaluation of the final checkpoint on 1,000 held-out puzzles
Hardware	Isambard-AI GH200 nodes; the seed-0 modular Countdown-3 rows at $f \geq 0.1$, the seed-0 5/10-epoch rows and the seed-0 344M solution-only ladder on one RTX 5090

Table 4: Model configurations. All models share the tokenizer and context length of Table 3.

model	layers	heads	d_{model}	d_{ff}	parameters
4L (30M)	4	12	768	2,048	30.3M
8L (59M)	8	12	768	2,048	58.6M
12L (87M)	12	12	768	2,048	87.0M
12L-h24 (344M)	12	24	1,536	4,096	343.8M

Table 5: Training runs behind the final figures, one row per arm (reasoning mode × trace corpus × model). Batch is the global batch in sequences (per-device × gradient accumulation; multiplied by the GPU count); steps are given for the full pool ($f=1$) at 1 epoch, and after a slash for 5 and 10 epochs; a run at fraction f takes $\lceil Nf/B \rceil \times$ epochs steps for a pool of N records, with a warm-up of 1% of its own steps. GPU-hours are W&B active wall time × GPUs (approximate; lower bounds for resumed runs). †The seed-0 8L maze rows trained on the 5M pool at raw fractions 0.1/0.2/0.4/0.6, the same sample counts as the 3M ladder; the search-augmented seed-0 row at $f=1$ trained under seed 1.

Sweep	Arm	Model	Batch	GPUs	Epochs	Steps at $f=1$	Runs	GPU-h
Maze 30 × 30	search-augmented, A* trace	4L (30M)	128 (64×1)	2	4	93,752	20	164
	search-augmented, A* trace	8L (59M)	128 (32×1)	4	4	93,752†	20	631
	search-augmented, A* trace	12L (87M)	128 (16×2)	4	4	93,752	20	488
	solution-only, A* trace	4L (30M)	128 (128×1)	1	4	93,752	20	51
	solution-only, A* trace	8L (59M)	128 (128×1)	1	4	93,752†	20	115
	solution-only, A* trace	12L (87M)	128 (64×1)	2	4	93,752	20	136
	swapped-trace, A* trace	8L (59M)	128 (32×1)	4	4	93,752	20	440
Countdown-4 (regular)	search-augmented, SoS trace	8L (59M)	128 (64×2)	1	1	156,250	45	74
	search-augmented, DFS trace	8L (59M)	128 (64×2)	1	1	156,250	45	75
	solution-only, optimal path	8L (59M)	128 (128×1)	1	1/5/10	156,250 / 781,250 / 1,562,500	55	83
	solution-only, optimal path	12L-h24 (344M)	256 (256×1)	1	1/5/10	78,125 / 390,625 / 781,250	55	105
Countdown-3 (modular)	search-augmented, DFS trace	8L (59M)	128 (128×1)	1	1	156,250	45	53
	search-augmented, DFS trace	12L-h24 (344M)	256 (256×1)	1	1	78,125	45	98
	solution-only, DFS trace	8L (59M)	128 (128×1)	1	1/5/10	156,250 / 781,250 / 1,562,500	55	83
	solution-only, DFS trace	12L-h24 (344M)	256 (256×1)	1	1/5/10	78,125 / 390,625 / 781,250	55	110
	swapped-trace, DFS trace	8L (59M)	128 (128×1)	1	1	156,250	30	46
Countdown-3 (modular)	search-augmented, SoS trace	8L (59M)	128 (128×1)	1	1	156,250	45	52

C SERIALISED COUNTDOWN TRAINING RECORDS

Records are printed as the model reads them, colour-coded by region: problem, query, search trace and solution: the shortest record of each corpus, and the shortest failed record of each SoS corpus.

Example 1. Regular Countdown, SoS trace, goal reached. 51 tokens.

```
70 63 75 32 query 100 reasoning close 70 63 75 32 create 75 - 70 5 close 63 32 5 create 63 +
32 95 close 5 95 create 5 + 95 100 solution plan 75 - 70 5 plan 63 + 32 95 plan 5 + 95 100 end
```

Example 2. Regular Countdown, SoS trace, search failed (empty solution). 14 tokens.

```
37 68 89 97 query 23 reasoning close 37 68 89 97 solution end
```

Example 3. Regular Countdown, DFS trace. 51 tokens.

```
5 5 33 33 query 76 reasoning close 5 5 33 33 create 5 + 5 10 close 33 33 10 create 33 + 33 66
close 10 66 create 10 + 66 76 solution plan 5 + 5 10 plan 33 + 33 66 plan 10 + 66 76 end
```

Example 4. Modular Countdown, DFS trace. 35 tokens.

```
186 454 651 query 290 reasoning close 186 454 651 create 186 + 454 640 close 640 651 create
640 + 651 290 solution plan 186 + 454 640 plan 640 + 651 290 end
```

Example 5. Modular Countdown, SoS trace, goal reached. 35 tokens.

```
186 454 651 query 82 reasoning close 186 454 651 create 454 + 651 104 close 186 104 create 186
- 104 82 solution plan 454 + 651 104 plan 186 - 104 82 end
```

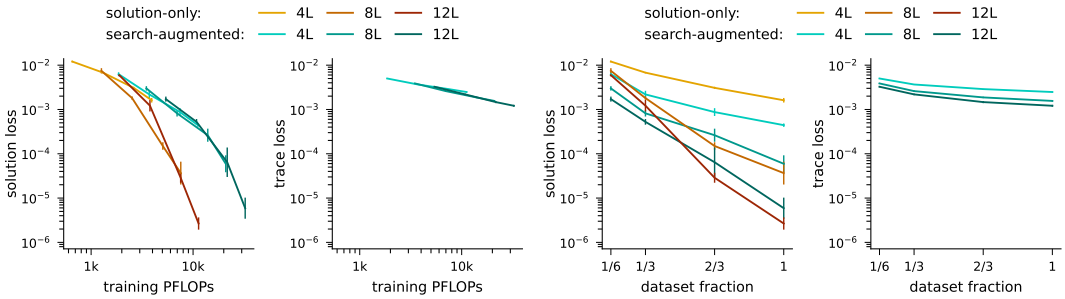
Example 6. Modular Countdown, SoS trace, search failed (empty solution). 12 tokens.

```
186 454 651 query 10 reasoning close 186 454 651 solution end
```

D TRAINING CURVES

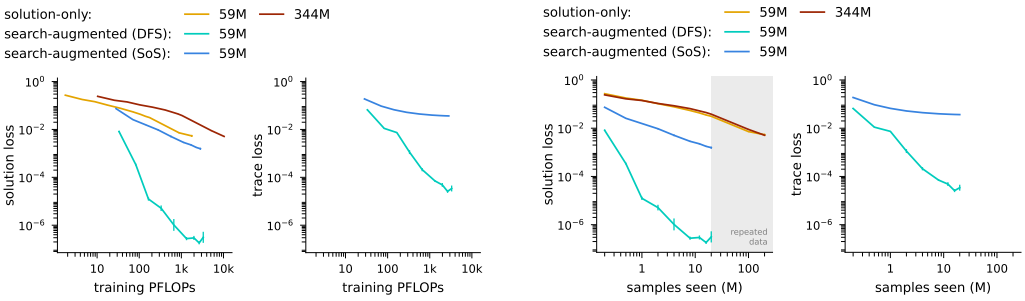
D.1 ADDITIONAL TRAINING CURVES

Figures 10–15 complement the loss and error figures of the main text with the second axis for every sweep. *Solution loss* and *trace loss* are the teacher-forced cross-entropies of each run's final checkpoint on the held-out split, averaged over the solution tokens (after `solution`) and over the trace tokens (between `reasoning` and `solution`) respectively, the same quantities as in the main-text loss figures. Markers are geometric means over the 5 seeds with ± 1 standard error in log space; the mixture sweeps have one run per point. Because a search-augmented model predicts the solution after a trace that already contains it, solution losses are not comparable across reasoning modes.



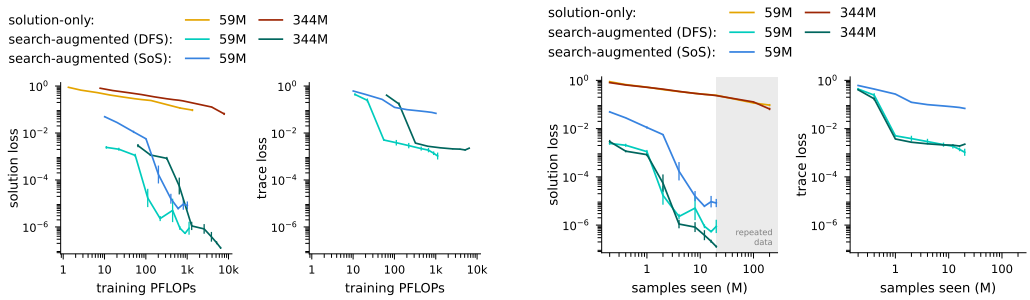
(a) versus training compute (b) versus data

Figure 10: Maze navigation: solution and trace loss for every model size and reasoning mode, 120 runs (the trace panel holds the 60 search-augmented runs).



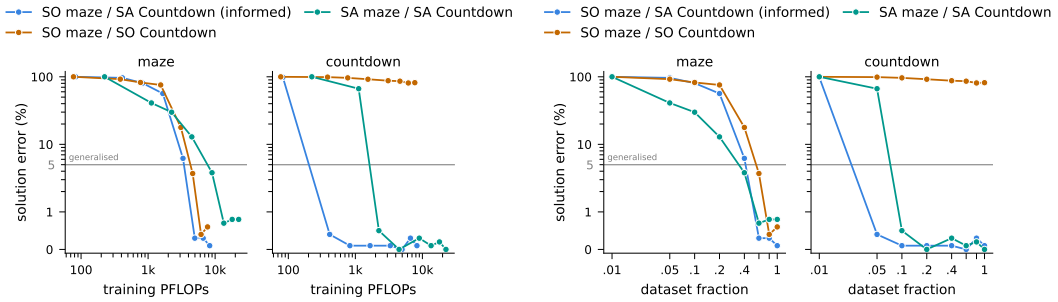
(a) versus training compute (b) versus data

Figure 11: Regular Countdown: 200 runs; the shaded region marks the solution-only runs that repeat the 20M-puzzle corpus for 5 and 10 epochs. No 344M model was trained with traces on this corpus.



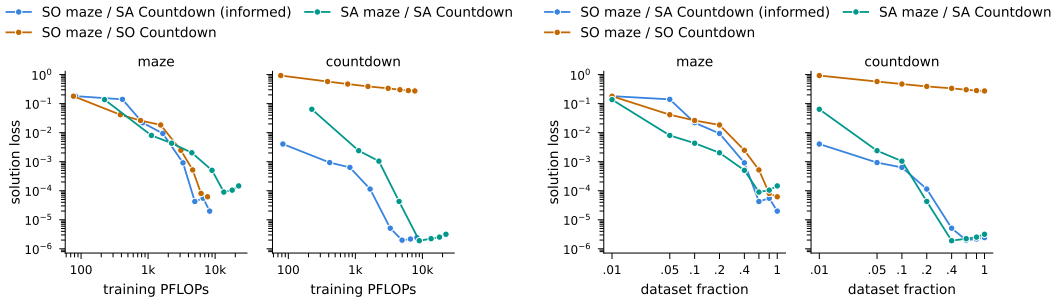
(a) versus training compute (b) versus data

Figure 12: Modular Countdown: 245 runs, including the 344M DFS-trace arm; the SoS-trace arm exists at 59M only.



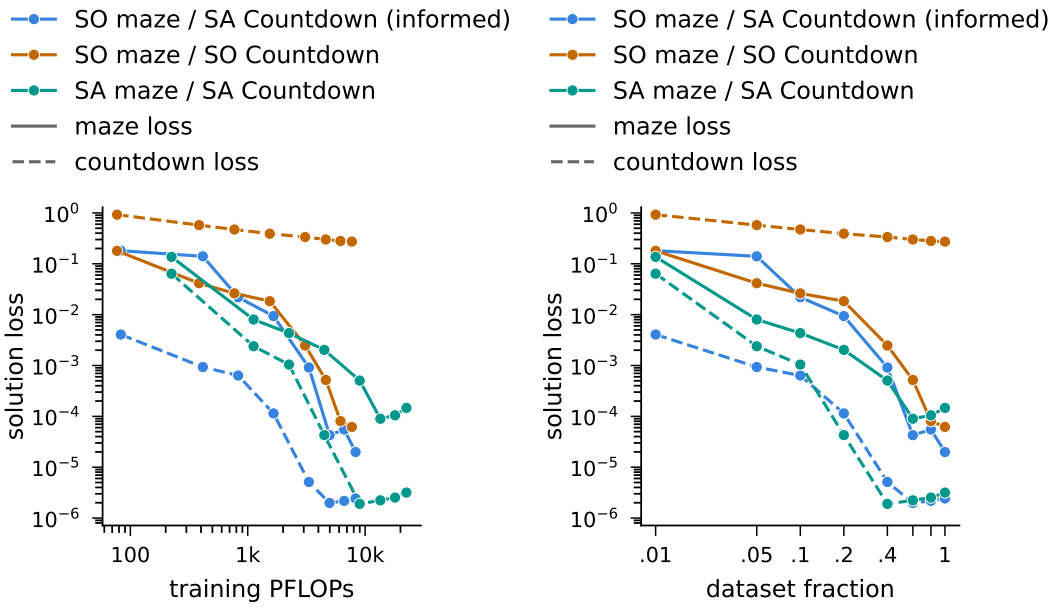
(a) versus training compute (b) versus data

Figure 13: Mixture training: solution error on each domain's held-out split (1,000 generations per domain) for the three supervision pairs, 24 runs.



(a) versus training compute (b) versus data

Figure 14: Mixture training: solution loss on each domain's held-out split, same runs. The mixture evaluation logs no trace loss.



(a) versus training compute

(b) versus data

Figure 15: Mixture training: the two domains' solution losses on one panel (solid: maze, dashed: Countdown). The per-domain evaluation records token-weighted means but not token counts, so no pooled loss can be formed.

D.2 TRAINING LOSS VERSUS OPTIMIZER STEPS

Figures 16–18 show the training loss of every run of the three sweeps against optimizer steps. The loss is the mean next-token cross-entropy over the tokens a run is trained on (trace and solution for search-augmented models, solution only for solution-only models), so its value is not comparable across reasoning modes. Curves are the logged loss (every 10 steps) averaged in log-spaced step bins (40 per decade); the 5 seeds of one dataset fraction share a colour.

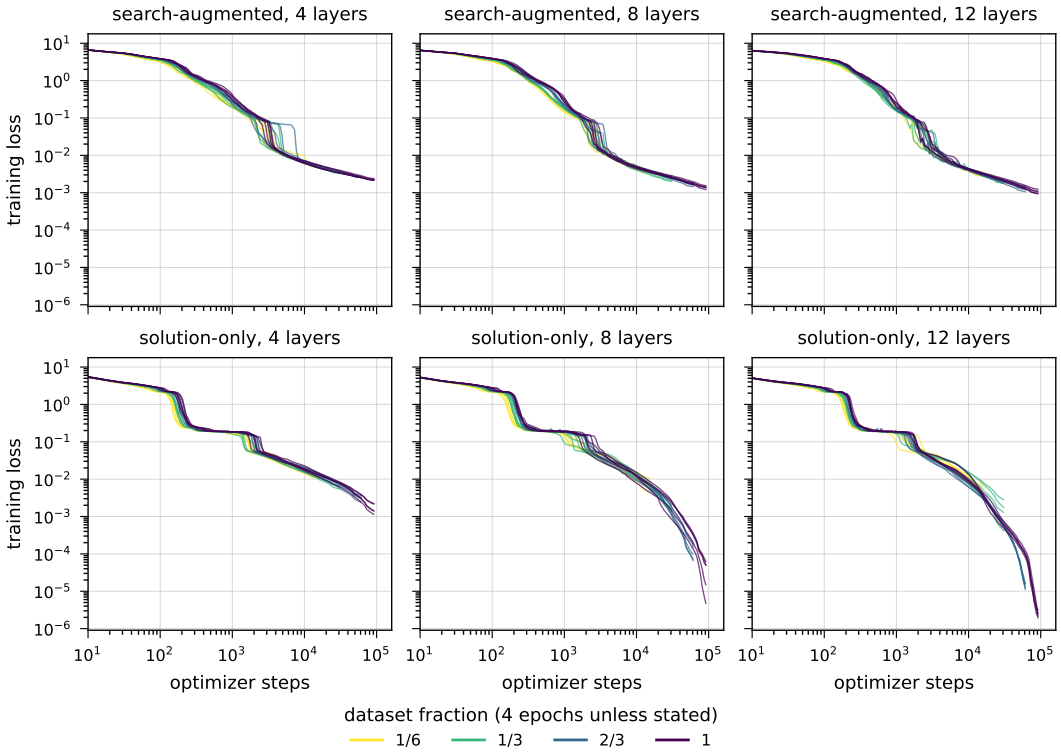


Figure 16: Maze navigation: 120 runs, four dataset fractions (0.5M, 1M, 2M and 3M mazes) × 5 seeds, four epochs each, for every model size and reasoning mode. The 8-layer seed-0 runs trained on an earlier 5M-record draw of the same generator at fractions 0.1–0.6, i.e. the same 0.5M–3M ladder.

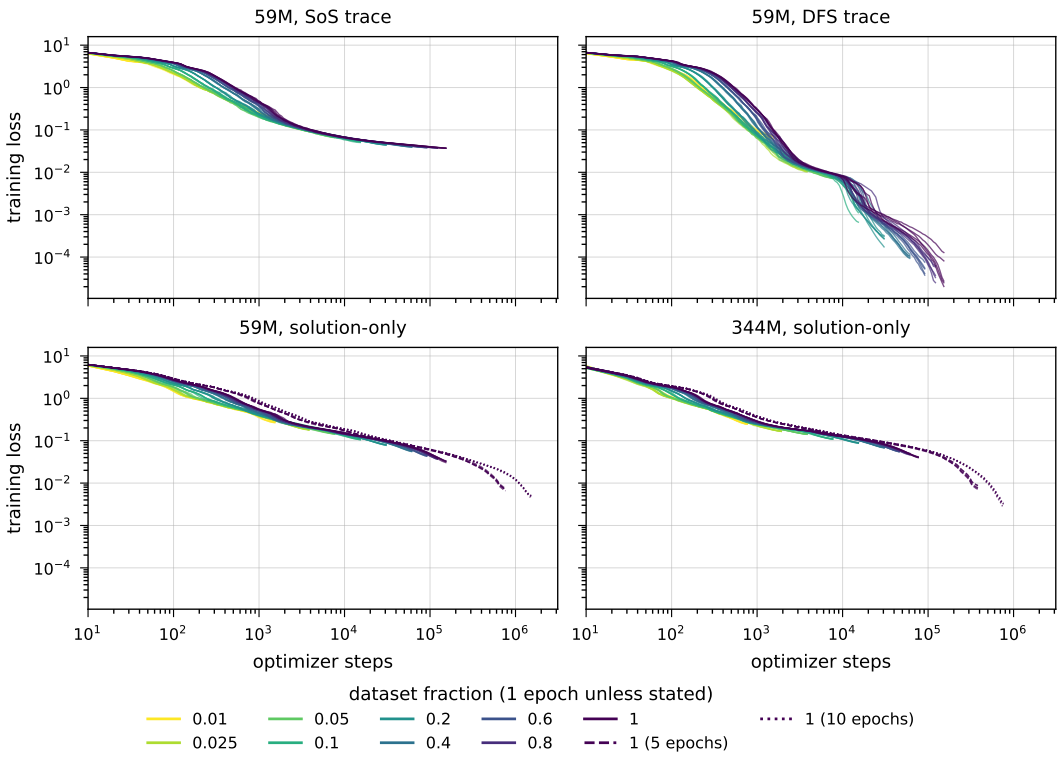


Figure 17: Regular Countdown: 200 runs, nine dataset fractions (0.01–1 of 20M puzzles) $\times$ 5 seeds, one epoch each; solution-only models were additionally trained on the full corpus for 5 (dashed) and 10 (dotted) epochs. No 344M model was trained with traces on this corpus.

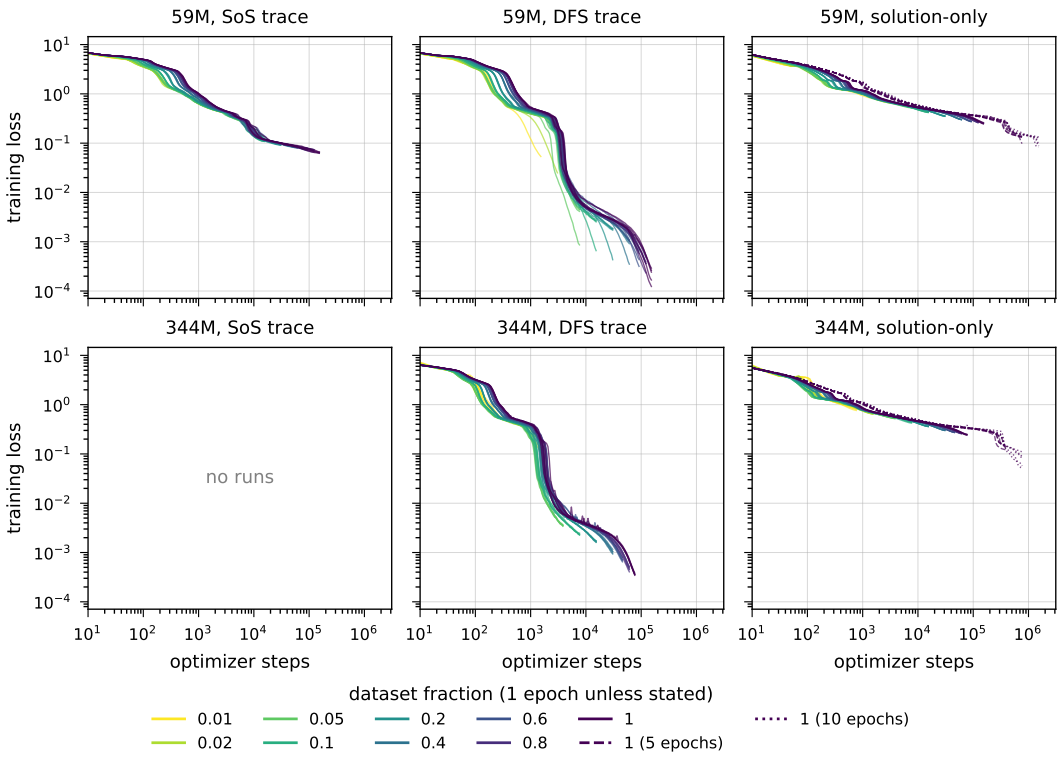


Figure 18: Modular Countdown: 245 runs, nine dataset fractions $\times$ 5 seeds, one epoch each, plus the 5- and 10-epoch full-corpus solution-only runs. The seed-0 59M runs at fractions ≥ 0.1 trained on an earlier, independently drawn 20M-puzzle corpus from the same generator.