Beyond Semantics: The Unreasonable Effectiveness of Reasonless Intermediate Tokens

Kaya Stechly[*]
SCAI, Arizona State University
kstechl@asu.edu

Karthik Valmeekam[*]
SCAI, Arizona State University
kvalmeek@asu.edu

Vardhan Palod[*]
SCAI, Arizona State University
vpalod@asu.edu

Atharva Gundawar
SCAI, Arizona State University
agundawa@asu.edu

Subbarao Kambhampati
SCAI, Arizona State University
rao@asu.edu

Abstract

Recent impressive results from large reasoning models have been interpreted as a triumph of Chain of Thought (CoT), and especially of the process of training on CoTs sampled from base LLMs in order to help find new reasoning patterns. In this paper, we critically examine that interpretation by investigating how the semantics of intermediate tokens—often anthropomorphized as "thoughts" or reasoning traces and which are claimed to display behaviors like backtracking, self-verification, and meta-cognition—actually influence model performance. We train transformer models on formally verifiable reasoning traces and solutions, constraining both intermediate steps and final outputs to align with those of a formal solver. By constructing a formal interpreter of the semantics of our problems and intended algorithm, we systematically evaluate not only solution accuracy but also the correctness of intermediate traces, thus allowing us to evaluate whether the latter causally influences the former. Our experiments involve training transformer models on traces and solutions generated by A* search. We notice that, despite significant improvements on the solution-only baseline, models trained on entirely correct traces still produce invalid reasoning traces when arriving at correct solutions. To further show that trace accuracy is only loosely connected to solution accuracy, we then train models on noisy, corrupted traces which have no relation to the specific problem each is paired with, and find that not only does performance remain largely consistent with models trained on correct data, but in some cases can improve upon it and generalize more robustly on out-of-distribution tasks. These results challenge the assumption that intermediate tokens or "Chains of Thought" reflect or induce predictable reasoning behaviors and caution against anthropomorphizing such outputs or over-interpreting them (despite their mostly correct forms) as evidence of human-like or algorithmic behaviors in language models.

[*] equal contribution

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: LAW 2025: Bridging Language, Agent, and World Models.

1 Introduction

Recent advances in general planning and problem solving have been spearheaded by so-called "Long Chain-of-Thought" models, most notably DeepSeek's R1 [17]. These transformer-based large language models are further post-trained using iterative fine-tuning and reinforcement learning methods. Following the now-standard teacher-forced pre-training, instruction fine-tuning, and preference alignment stages, they undergo additional training on reasoning tasks: at each step, the model is presented with a question; it generates a sequence of intermediate tokens (colloquially or perhaps fancifully called a "Chain of Thought" or "reasoning trace"); and it ends it with a specially delimited answer sequence. After verification of this answer sequence by a formal system, the model's parameters are updated so that it is more likely to output sequences that end in correct answers and less likely to output those that end in incorrect answers.

While (typically) no optimization pressure is applied to the intermediate tokens [4, 56], empirically it has been observed that language models perform better on many domains if they output such tokens first [34, 46, 54, 20, 16, 17, 37, 32, 28]. While the fact of the performance increase is well-known, the reasons for it are less clear. Previous work has often framed it in anthropomorphic terms, claiming that these models are "thinking" before outputting their answers [34, 14, 17, 49, 56, 7]. Simultaneously, the process of performing more auto-regressive forward passes before outputting the final answer has been credited as an instance of *inference-time scaling* – that is, these models are assumed to be doing problem-adaptive computation.

Famously, DeepSeek's R1 paper claimed that one of the most impressive observed behaviors of their trained models was the so-called "aha" moment: as part of the chain of thought it was producing in order to answer some question, the model output the token "aha", seeming to indicate that it had come upon a sudden realization. Interpreting this token as meaningful to the end user requires making an additional assumption that has thus far been brushed to the side in discussions of how long CoT models function and what they do – that the derivational traces they produce are semantically meaningful to the end user in the same way that the traces they were trained on were. While the traces certainly seem to help the LLM performance (and may well have mechanistic interpretability properties [6]), it is not clear (beyond anecdotal evidence) that they have end-user interpretability or semantics.

For R1 and similar large models, this is nearly impossible to check. The intermediate tokens that massive pre-trained and post-RL'd models produce meander for dozens of pages, are written wholly in ambiguous and polysemantic natural language, and – perhaps much worse – are the result of long, opaque training processes on data that we have no access to and cannot compare against.

In this paper, we shed some light on the question of whether intermediate traces are semantically meaningful. Following previous work that elucidated important functional aspects of large scale models through controlled small scale experiments [45, 38, 55] and working within a sort of "model organism" paradigm, we focus on fully controlled, open, and replicable models trained from scratch. Our models are trained on a simple and well-understood shortest path planning problem for randomly generated mazes, with our training runs including varying kinds of intermediate traces – from none to ones generated by the classic A* algorithm to noisy and irrelevant ones. This maze solving domain is not only well-understood as a classical computer science problem, but has also grown to be well-studied domain for trace-augmented transformer training that led to reasoning models [27, 41, 33, 50].

We approach the problem of understanding intermediate token semantics from three major novel angles, performing empirical evaluations on models we train on small planning tasks. First, we construct a validator for A* execution traces and use it to validate and compare trace accuracy to solution accuracy, finding only a loose correlation between the two. Then, we train half billion parameter Qwen models on none, correct, and deliberately irrelevant traces. We present a dataset manipulation involving swapping the intermediate tokens of two different problem instances, that – despite the fact that it removes all specific-problem-relevant semantics – leads to trained models that perform better on both in and out of distribution tasks. We also show that applying post-training methods like GRPO [40, 17] can further boost the accuracy of those models while consistently producing incorrect intermediate tokens. We argue that, if performance is the goal, assuming human-like or algorithm-interpretable trace semantics is not only misleading but potentially unnecessary.

2 Related Work

Post-Training for Reasoning - Recent progress in improving the reasoning capabilities of Large Language Models (LLMs) has been driven by methods that train models not only on correct answers, but also on "reasoning traces" that lead to those answers [53, 28, 32, 29, 25, 52, 17, 42, 17, 3, 51]. Whether this is achieved via supervised fine-tuning on trace-augmented datasets or via reinforcement learning techniques like Group Relative Policy Optimization (GRPO), the end result is the same: models "think" for varying amounts of time by outputting additional tokens before outputting their final answers, and performance measures improve.

These methods are often paired with claims about how and why they work, which lean on anthropomorphic framings of model "thinking", which seem to assume that final answer correctness somehow requires intermediate sequence correctness. These claims are even presented alongside evidence that seems to directly contradict them. The training procedure underlying DeepSeek's R1-Zero is instructive[17]: almost all the of the performance improvement can be recovered by reinforcing the model's behavior based entirely on the correctness of the final answer, without any reference to the content of the trace – incorrect traces that happen to lead to correct answers are treated exactly the same as correct ones. As they hint, rewarding intermediate correctness might even be counterproductive. In the current work, we take this idea seriously and push it even further: showing that rewarding the semantically incorrect thing can actually lead to better performance than rewarding coherent and correct traces.

Evaluating Traces - Previous work on evaluating the relationship between trace correctness and final answer correctness has primarily focused on large, pre-trained models, and has gone under the name of Chain-of-Thought faithfulness. These evaluations have claimed that intermediate steps, despite appearing coherent, are not reliably correct [44, 26, 8, 10, 2, 4]. However, the nature of natural language makes these evaluations questionable. Within the domain of math problems, there is no established ground truth semantics for what a correct reasoning process looks like in natural language, and so previous work either relies on messy manual evaluations which typically require the researcher to read in some amount of intent into the model's completions, or automated evaluations that use additional (unverified) language models to noisily pick out potential reasoning errors.

Training Transformers on Traces - Prior efforts have trained models from scratch to mimic search algorithms like A*, Breadth-First Search (BFS), and Monte Carlo Tree Search for tasks in pathfinding, arithmetic, and general problem-solving [27, 15, 48], as well as the DPLL procedure for Boolean SAT problems [35]. However, while these studies train on traces of formal procedures, they do not explicitly analyze the correctness of the intermediate traces themselves. In fact, some works have even employed semantically invalid traces—such as truncated A* derivations in Dualformer [41]—without examining how these deviations affect model performance. By contrast, our work is the first to rigorously evaluate trace correctness, allowing us to directly investigate the relationship between the semantics of intermediate tokens and the generated solutions.

3 Background

Though recently popularized by Large Reasoning Models, especially DeepSeek's R1 [56], training on intermediate traces in order to improve transformer performance dates back to at least GPT-2 [34] and has been extended and analyzed from many angles [27, 53, 15, 41, 23, 31, 13] While these papers demonstrated ways to improve final answer accuracy, they neither evaluate the trace accuracy nor do they explicitly attempt to train on incorrect or irrelevant traces.

Thus, while they do show that accuracy increases, they leave open the question of whether that accuracy increase actually stems from the additional semantic information in the trace. In many cases, especially with pre-trained models, it is nearly impossible to formally verify reasoning traces, due to the ambiguity of natural language and the lack of a clear ground truth.[2] However, for small, well-scoped formal domains like the gridworld path planning domain used in [27, 41] and this paper, and with carefully from-scratch trained models on those domains, we have the ability to check whether generated traces follow the exact semantics enforced in the training data and causally predict the final solutions that the model outputs.

[2] This in turn induces a Rorschach test-like behavior in the end users who overload semantics on specific phrases like "aha" or "let me think".

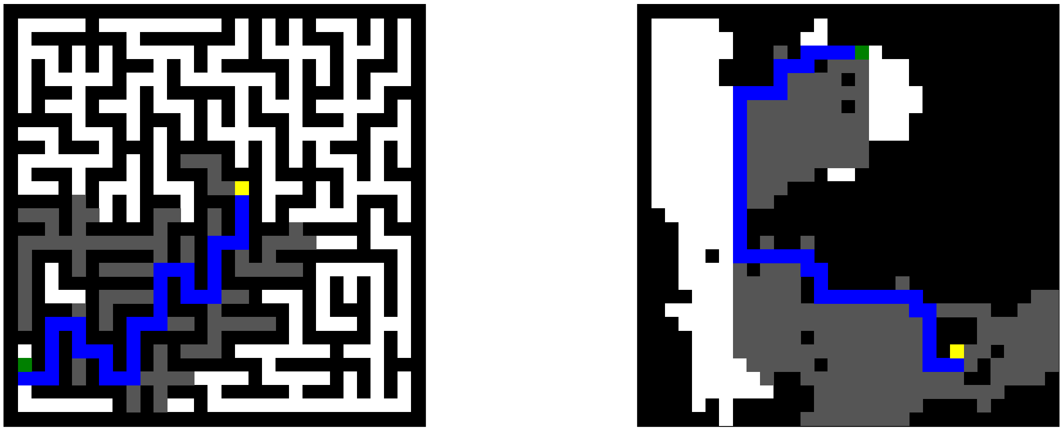


Figure 1: Examples of mazes. The left is generated by Wilson's algorithm and is used for model training. The right is generated by the Drunkard's Walk algorithm and used to evaluate models as an out of distribution task. The goal is represented by a green square and the start state by a yellow square. Black squares represent impassable walls. Blue squares represent steps along the optimal path (as found by A*). Gray squares are squares that were explored by A* but are not along the optimal path. White squares are unexplored traversable squares.

3.1 The Maze Pathfinding Domain

We consider a standard grid-based path-finding domain. The task is to find a legal path between a given start cell and goal cell in a 30×30 grid. Every cell of this grid is either free (traversable) or a wall (impassable). The agent begins at the start state, and at every state may take one of four actions: go up, go down, go left, or go right. The transformer is given a full description of this problem (in token format – we follow the formulation used by [27] and [41]) and must output as its final answer a plan, which consists of a sequence of actions. A plan is considered correct if it executable – that is, every action it presents moves the agent from a free cell to an adjacent free cell – and its final action results in the agent being at the goal cell.

In order to understand out of distribution performance, we generate navigation problems using diverse generation algorithms, resulting in varied structural patterns and exploration dynamics. This enables systematic out-of-distribution (OOD) evaluation by testing models on maze types unseen during training – which was all done on mazes generated with Wilson's algorithm. These generation algorithms can be sorted into two major categories: 1) algorithms that do not permit cycles and sample over the spanning trees of the 30×30 grid and 2) algorithms that permit loops and create noisy, less-structured dungeon or cave-like instances. For all algorithms except SearchFormer's, which has its own start and goal generation loop, we sample a legal (start, goal) pair after maze generation.

Acyclic Maze Generation

1. **Wilson's algorithm:** This is the algorithm that we use to generate mazes for training models. Wilson's algorithm generates uniform random mazes by performing loop-erased random walks from unvisited cells until they connect to the current maze [47]. Each walk removes any loops it creates, ensuring a valid tree structure. This process continues until all cells are included, producing a uniform sample from the space of all possible spanning trees of the 30×30 graph.
2. **Kruskal's algorithm:** Kruskal's algorithm, originally proposed for finding a minimum spanning forest of an undirected edge-weighted graph [24], generates mazes by treating each cell as a node and randomly removing walls between unconnected regions, using a union–find structure to avoid cycles. This results in a fully connected maze without loops, though the maze distribution is not perfectly uniform. The method produces mazes biased towards short local connections and dead ends.
3. **Randomized Depth-First Search algorithm:** The randomized depth-first search (DFS) or recursive backtracker algorithm generates mazes by carving a path forward until reaching a

dead-end [43]. When it hits a dead-end (no unvisited neighbors), it backtracks until it finds a new direction to explore, repeating until all cells are visited and connected into a complete maze. Depth-first search is biased towards generating mazes with low branching factors and many long corridors.

Cave Generation

4. **Drunkard's Walk:** We implement a version of the "Drunkard's Walk" algorithm, as described by [22], and originally used for procedurally generating dungeons for top-down two-dimensional video games. Starting from a grid of solid walls, a random walk is performed, carving out the current cell on every step. The walk continues until a predefined number or percentage of floor tiles has been dug out. This method preserves cycles, producing cave-like structures with open chambers and looping corridors. The output space includes grid states unreachable by perfect acyclic maze generators.
5. **SearchFormer style generation** We also implement the random generation algorithm used in the SearchFormer paper [27], though we use it for evaluation rather than training. Tasks are generated by exhaustive rejection sampling: first randomly select a number between 30% and 50%. Then select that percentage of cells to be wall cells. Randomly choose a start and goal location and execute A* to find an optimal plan. Reject unsolvable, too easy, or duplicate instances and resample. These instances also allow for loops and so are also out of distribution for our models.

3.2 The A* Search Algorithm

A* is a classic best-first graph–search procedure that combines the uniform-cost guarantee of Dijkstra's algorithm [12] with domain-specific heuristics to focus exploration on promising states, originally introduced to compute minimum-cost paths in state-space graphs [19].

The algorithm maintains an *open list* (a priority queue) keyed by $f(n) = g(n) + h(n)$, where $g(n)$ is the exact cost from the start and $h(n)$ is a heuristic estimate to the goal, and also maintains a *closed list* of already visited nodes. It repeatedly pops the open list node with the smallest f; if this is the goal, it reconstructs the path that lead to this node and this is returned as the final plan. Otherwise, it generates child nodes (in our case, traversable neighbor cells) and calculates their g and f values. For each node, it either inserts it into the open list or – if the node is already in the list – updates its g value if the new value is lower. The popped node is added to the closed list to prevent re-expansion.

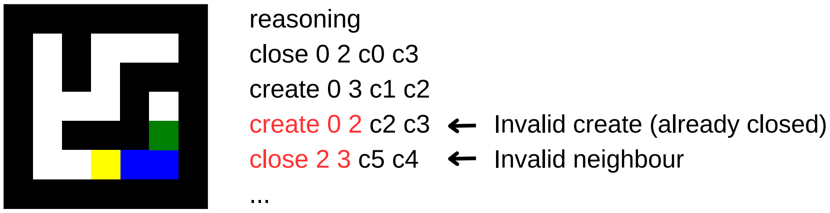
The effectiveness of A* is dependent on the heuristic it is implemented with. For solvable graph search problems like the ones featured in this paper, any consistent ($h(n) \leq c(n, n') + h(n')$ for all neighboring n') heuristic will guarantee that the plan returned is not only satisficing but optimal [36].

For the maze path planning problems we examine in this paper, we use the very standard Manhattan heuristic $h(n) = |x_n - x_g| + |y_n - y_g|$ which computes the sum of horizontal and vertical displacements between a cell and the goal. On a 2-D grid with only orthogonal, unit-cost movement, this heuristic is consistent, ensuring A* returns an optimal path.

Finally, following SearchFormer and Stream of Search, we modify the A* implementation to output a linearized execution trace [15, 27]. That is, whenever the algorithm creates a child node and adds it to the open list, it prints `create x y cA cB` and when it closes a node and adds it to the closed list, it prints `close x y cA cB`. Here, 'A' in cA represents the exact cost from the start state to the node (i.e., the $g(n)$ value) and 'B' in cB represents the heuristic estimate from that node to the goal state (i.e., the $h(n)$ value). Similar to Searchformer notation [27], we use the prefix "c" to differentiate between the node co-ordinates and its cost estimations. In the next section, we construct an A* validator that reverses this process – it takes in a linearized trace and attempts to simulate the corresponding open and closed list operations to check if they are valid with respect to the semantics of this implementation.

4 Validating Traces and Solutions

While previous work evaluated the final accuracy of trace-trained models, it did not evaluate the traces themselves. For large, production ready RL-post-trained models like DeepSeek's R1, this is



```
reasoning
close 0 2 c0 c3
create 0 3 c1 c2
create 0 2 c2 c3  ← Invalid create (already closed)
close 2 3 c5 c4   ← Invalid neighbour
...
```

Figure 2: Trace validation procedure. Our A* validator runs through the model's output stream sequentially. Assuming no parsing errors, it will flag a trace as invalid if at some point it contains an invalid action. The left bottom corner is $(0, 0)$.

practically impossible. For even a simple query, the model produces dozens of pages of convoluted and meandering output before arriving at an answer, and this output is all in natural language, which makes it very easy to read multiple equally valid interpretations into it.

To truly tell whether the traces that were trained on helped in the expected way, we need a formal way of validating their correctness. By training models on traces produced by a well-known algorithm with well-known semantics, it is possible to check whether the model's emulations of the algorithm's execution trace are correct.

We construct a formal verifier for A* traces. The format for these traces follows [27], and is described in more detail in Section 3. Essentially, our validator consumes the generated trace and simulates the operations proposed in that trace on open and closed lists. It runs through the generated trace sequentially, parsing each `action x y cA cB` sequence as an operation and using it to update its open and closed list. It marks a trace as valid if it can correctly execute this procedure until it closes the goal node. Errors in execution can be one of the following:

- **Parsing Error**: a substring is malformed and does not parse into either a create or a close action with the correct arguments.
- **Invalid Neighbor**: the current create action is attempting to create an illegal child, either referencing a wall cell or a cell that is not adjacent to the last closed node.
- **Already Closed**: the current create action is attempting to close an already closed node.
- **Not in Open List**: the current close action is referencing a node that is not in the open list.
- **Not Lowest f-value**: the current close action is attempting to close a node when there is at least one other node in the open list with a lower f-value.
- **Goal Not Reached**: after the entire sequence was processed, the goal node was not in the closed list, and so the reconstruction step cannot proceed.

With this verifier in hand, we can now distinguish between plan validity and trace validity for models trained on this kind of dataset. To construct our training sets, we generate 50,000 mazes using Wilson's algorithm, and randomly select a start and goal cell. Then, we use A* with the Manhattan distance heuristic to find an optimal plan for each maze as well as to produce a trace that is saved with each datapoint.

We modify the architecture of the Qwen2.5 0.5B [39] to support a vocabulary of exactly 944 different tokens (which reduces the parameter count to about 380 million from 500 million), randomly initialize the model, and then train it for 85,000 training steps with a batch size of 8 on two NVIDIA H100s. The model has a context length of 32,000 tokens to support the long lengths of intermediate token generation. Note that we are training empty transformer models from scratch instead of fine-tuning pre-trained models. (Our other experiments later in the paper also use this architecture, but train on different datasets, from solution-only through to irrelevant and noisy traces. All code and data will be made public.) We test this model trained on Wilson mazes on a thousand instances of mazes generated by Wilson, Kruskal, DFS, SF-Style and Drunkard approaches, evaluating the solution accuracy as well as the trace validity. We present these results as confusion matrices in Figure 3, with each domain represented by a separate matrix. These results break down the correlation between model accuracy and trace validity. As can be seen from the results, trace accuracy is not a perfect predictor of plan accuracy. In fact, as can be seen from the diagonal entries, the model can produce

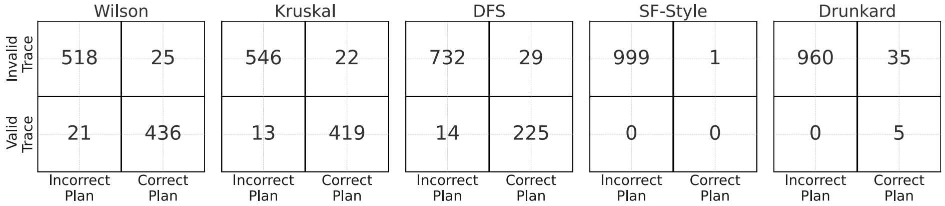


Figure 3: Plan versus trace validity for the model trained on correct traces, measured across domains. Wilson = generated by Wilson's algorithm, Kruskal = mazes generated by Kruskal's algorithm, DFS = mazes generated by Depth-First Search, SF-Style = instances generated in the SearchFormer Style, Drunkard = instances generated using the Drunkard's algorithm.

valid traces and then continue on to produce an incorrect plan or produce invalid traces and yet end up at a correct plan[3].

5 Training with Traces: Does Meaning Matter?

If plan and trace validity are only loosely connected for models trained on the A* trace dataset, then perhaps the validity of the trace isn't as important to the performance increase as previously believed. To test this empirically, we construct a second training dataset called Swap, which we build by randomly permuting reasoning traces between problems.

This dataset consists of the exact same problems as the original Trace 50,000, but problem 1's trace will be given to, say, problem 4; problem 4's will be given to problem 7; and so forth. In other words, while the traces continue to have the right form and some generic domain information, they no longer have any connection to the specific problems they are associated with. Training examples consist of a start and goal state, a maze definition, an A* trace for searching for the shortest path across a totally unrelated maze from a different start and goal state, and the correct solution plan for the original maze.

What we find is that our most competent model not only maintains performance on the in-distribution test set, but it generalizes better than the other maze distributions we test! All despite the lack of algorithmically valid semantics in the trained upon and generated traces.

For these experiments, we continue to use the same model architecture described in the previous section, varying the datasets we train on to see how they affect performance – even as they further corrupt or completely destroy the correctness of the traces. For best results, we performed hyperparameter optimization (via Optuna [1]). We provide additional details on hyperparameters and initializations in the Appendix.

The most basic training run is the standard solution-only baseline, where the model is trained on just solutions without any derivational traces. The next baseline, following previous work [27, 41, 15] is training the model with A* generated traces, teacher-forcing during training to make it output intermediate tokens before the final solution. These are the models discussed in the previous section. Finally, we use the same underlying training data and distribution, but modify it by corrupting the traces. Our trace corruption process is very simple: we randomly permute which problem is associated with which traces – so, for example, the third problem might have the 5th problem's trace, which is an execution trace of A* on an unrelated maze with unrelated start and goal states.

The problems in our training data are all generated by Wilson's algorithm. For our test sets we generate data with several maze generation algorithms (as described in Section 3), including Wilson's algorithm, to get both in and out of distribution data. Our training data consists of 50k samples, while our test sets each contain 1k.

Unintuitively, as seen in Table 1, the best model in both in and out of distribution test sets turns out to be the model trained on swapped (incorrect) traces! We see that the swapped model has a 0% trace validity – as it has been trained to output well-structured but problem-irrelevant traces

[3] In the appendix, we also include a similar set of results for the models trained by [27], which show similar trends.

Table 1: Performance of Swap, A* Trace, and Solution-Only Models across maze distributions for models trained on 50k datapoints. "Plan Val." = Plan Validity, "Trace Val." = Trace Validity within Valid Plans

Test Set	Soln. Only	Regular A* traces		Swapped A* traces	
	Plan Val.	Plan Val.	Trace Val.	Plan Val.	Trace Val.
Wilson	4.0%	50.1%	95.2%	51.6%	0%
Kruskal	2.1%	49.7%	96.2%	51.9%	0%
DFS	2.8%	30.8%	82.1%	41.7%	0%
Drunkard	0.0%	2.5%	4.0%	26.0%	0%
Searchformer-style	0.0%	0%	0%	0.2%	0%

in response to every problem – but nevertheless performs noticeably better than both the correct trace and solution-only baselines. An interesting point to note is the performance difference on out-of-distribution datasets. While most of the performance differences are within a few percentage points, and in-distribution testing results in near identical performance, on the Drunkard dataset the swapped model is 10 times better than the original model, giving 26% to the correct trace model's 2.6%, and on the DFS maze set, it reaches 41.7% to the original model's 30.8%.

If intermediate tokens improve accuracy because they teach the model a given reasoning procedure, then we should expect their influence on performance to fluctuate exactly with their connection to the problem. However, we find that this is not always the case – in fact, intermediate token sequences that have almost nothing to do with the problem at hand can provide a significantly higher performance boost (and which, counterintuitively might even generalize better) than well-grounded semantically meaningful execution traces, thus throwing doubt on the seemingly wide-spread intuition that the effectiveness of traces stems from allowing the transformer to perform structured, interpretable, and algorithmic procedures.

5.1 Training on more data

To check the influence of larger training sets on performance, we trained models on datasets containing 500k data points for 250k steps. We observe that the swapped-trace model achieves much better performance when compared with the regular trace model in both in-distribution and out-of-distribution test sets, and both models substantially outperform the solution-only baseline. This consistency across training regimes reinforces our finding that intermediate token correctness is not required for performance benefits typically attributed to reasoning traces.

Table 2: Performance of Swap, A* Trace, and Solution-Only Models across maze distributions for models trained on 500k datapoints. "Plan Val." = Plan Validity, "Trace Val." = Trace Validity within Valid Plans

Test Set	Soln. Only	Regular A* traces		Swapped A* traces	
	Plan Val.	Plan Val.	Trace Val.	Plan Val.	Trace Val.
Wilson	10.0%	79.9%	97.1%	83.3%	0%
Kruskal	6.9%	83.1%	95.6%	85.0%	0%
DFS	0.0%	54.0%	87.41%	52.6%	0%
Drunkard	0.1%	0.0%	0.0%	11.7%	0%
Searchformer-style	0.0%	0.0%	0.0%	0.2%	0%

5.2 Does GRPO increase the semantic correctness of the intermediate tokens?

Considering that claims anthropomorphizing the relationship between intermediate tokens and the solution (such as the "aha" moment) [34, 14, 17, 49, 56] originate from models that are post-trained with methods such as GRPO, we post-train our base models with GRPO to ascertain whether this influences the trends seen in the previous section. We generate a separate dataset with 10k datapoints and use the models that were trained on 500k datapoints for 250k training steps as the base models. We provide additional hyperparameter details in Appendix A.1

Our findings (as shown in Table 3) indicate that although GRPO enhances the overall accuracies for both the regular and the swapped models, the swapped model consistently outperforms and demonstrates greater improvement than the regular model, despite its trace validity remaining at a consistent 0%! Even for the correct model, trace validity slightly decreases with post-training. This, indeed, shouldn't be surprising because GRPO typically doesn't put any optimization pressure on the intermediate tokens. These observations provide further evidence that the semantics of intermediate tokens do not bear significance for performance.

Table 3: Performance of Swap and A* Trace Models with GRPO. Accuracy represents the percentage of valid plans out of total instances (500). Trace Validity represents the percentage of valid traces within valid plans. RM: Regular Model, SM: Swapped Model, TV: Trace Validity.

Metric	Step 10	Step 20	Step 30	Step 40	Step 50	Step 60	Step 70
RM Accuracy (%)	80.0	82.0	85.0	87.0	87.0	87.0	88.0
SM Accuracy (%)	83.6	85.2	85.8	87.0	87.6	90.4	**92.2**
RM TV (%)	97.5	97.6	96.5	96.6	96.6	96.6	96.6
SM TV (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5.3 Intermediate Tokens Don't Need to be Thoughts

Our results hint that the impact of trace content on performance and the legibility of that content have been somewhat conflated – if all we care about is increasing the accuracy and capability of a model, enforcing human readability may be counterproductive, a lesson also mentioned in the R1 paper [56]. Furthermore, examining traces produced by a model – though they may look right at first glance – is not necessarily informative if those traces are not predictive of the model's final answers.

Of course, if trace semantics don't matter, then the question immediately arises: why does generating intermediate tokens increase accuracy at all? We speculate that what is helping is finding the right prompt augmentation. That is, for a given task prompt T, there exists a prompt augmentation PA which boosts the LLM's performance on that task:

$$\exists\, PA \text{ s.t. } \mathbb{P}\big(\mathrm{Sol}\big(\mathrm{LLM}(T + PA),\, T\big)\big) > \mathbb{P}\big(\mathrm{Sol}(\mathrm{LLM}(T),\, T)\big)$$

Here $\mathrm{Sol}(y, T)$ indicates that y solves T, and $\mathrm{LLM}(x)$ is the model's completion for input x. The central challenge then is to learn the Skolem function

$$\mathrm{PA} = f_\theta(T, \mathrm{LLM}),$$

that maps each task to an effective augmentation. This can be accomplished through modifying the model itself to inherently and automatically augment prompts, as is the case in models that first generate long chains of intermediate tokens before their final answers. Crucially, prompt augmentations have no need to be human-interpretable. In fact, we see results that back this up in the adversarial prompting literature, where effective jailbreaks can be effected by augmenting prompts with human-uninterpretable strings [57, 9, 30, 18] or modifying them with random syntactic permutations, capitalizations, and shufflings [21].

6 Conclusion

In this paper, we challenged the prevailing narrative that intermediate tokens or "Chains of Thought" generated by Large Reasoning Models like DeepSeek's R1 are interpretable, semantically valid sequences with predictable effects on the model's behavior. As we don't have access to any frontier LLM's training data or even exact training procedure, and since the traces these models output are in multiply-interpretable natural language without a concrete ground truth, we designed a series of experiments building on previous smaller model reasoning work – mainly Searchformer and Stream of Search [15, 27] – and constructed an A* trace validator, finding that there is only a loose correlation between the correctness of the trace and the correctness of the output plan. We then trained additional models on noisy or irrelevant traces and found that there are (nonsensical) trace formats that nevertheless maintain or even increase the model's performance – all despite them being much less informative or connected to the problem at hand. Finally, we argue that, if the goal is to increase model performance, enforcing trace semantics is unnecessary [5] and potentially very misleading. All together, our counter-intuitive results demonstrate ways in which common interpretations of the intermediate tokens produced by Large Reasoning Models may be anthropomorphizations or simplifications. While the traces certainly seem to help the LLM performance (and may well have mechanistic interpretability [6]), there is little reason to believe that they have end-user interpretability/semantics.

Acknowledgments and Disclosure of Funding

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

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. *arXiv preprint arXiv:2503.08679*, 2025.

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

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

[5] Siddhant Bhambri, Upasana Biswas, and Subbarao Kambhampati. Do cognitively interpretable reasoning traces improve llm performance?, 2025.

[6] Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter?, 2025.

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

[8] Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

[9] Valeriia Cherepanova and James Zou. Talking nonsense: Probing large language models' understanding of adversarial gibberish inputs, 2024.

[10] James Chua and Owain Evans. Are deepseek r1 and other reasoning models more faithful? In *ICLR 2025 Workshop on Foundation Models in the Wild*.

[11] Joseph Culberson. Sokoban is pspace-complete. 1997.

[12] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[13] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36:70757–70798, 2023.

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

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

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

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

[18] William Hackett, Lewis Birch, Stefan Trawicki, Neeraj Suri, and Peter Garraghan. Bypassing prompt injection and jailbreak detection in llm guardrails, 2025.

[19] 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.

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

[21] John Hughes, Sara Price, Aengus Lynch, Rylan Schaeffer, Fazl Barez, Sanmi Koyejo, Henry Sleight, Erik Jones, Ethan Perez, and Mrinank Sharma. Best-of-n jailbreaking. *arXiv preprint arXiv:2412.03556*, 2024.

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

[23] Juno Kim and Taiji Suzuki. Transformers provably solve parity efficiently with chain of thought. *arXiv preprint arXiv:2410.08633*, 2024.

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

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

[26] Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

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

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

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

[30] Yue Liu, Xiaoxin He, Miao Xiong, Jinlan Fu, Shumin Deng, and Bryan Hooi. Flipattack: Jailbreak llms via flipping. OpenReview pre-print, submitted to ICLR 2025, 2024.

[31] William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

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

[33] Niklas Nolte, Ouail Kitouni, Adina Williams, Mike Rabbat, and Mark Ibrahim. Transformers can navigate mazes with multi-step prediction. *arXiv preprint arXiv:2412.05117*, 2024.

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

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

[36] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley, 1984.

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

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

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

[40] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

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

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

[44] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

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

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

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

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

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

[50] Yongjing Yin, Junran Ding, Kai Song, and Yue Zhang. Semformer: Transformer language models with semantic planning. *arXiv preprint arXiv:2409.11143*, 2024.

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

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

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

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

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

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

[57] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.

A Appendix

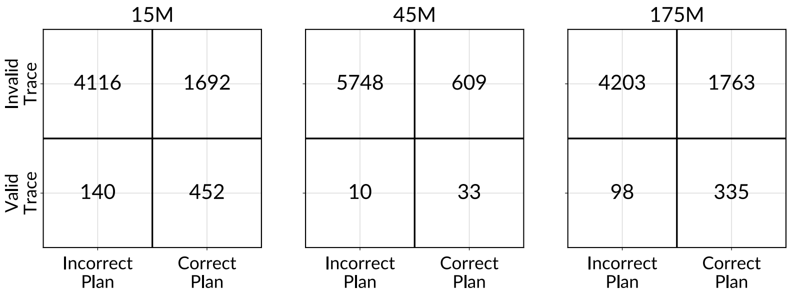
A.1 Additional experiment details

For all our experiments, we trained the Qwen-2.5-0.5B decoder only models. We used a custom tokenizer with domain specific vocabulary which reduced the model parameters to around 380M. We optimized with AdamW (β_1=0.9, β_2=0.999) and applied a weight decay of 0.1528, a peak learning rate of 2.2758e-4, and 100 warm-up steps, all under bf16 precision. We train the models for 95k training steps. All randomness was controlled with fixed seeds. The training dataset size is 50k datapoints unless specified otherwise.

For GRPO, we use 16 as the sample size, 256 as the batch size and 0.01 as the entropy coefficient for both the models.

A.2 Validating Traces and Solutions of Searchformer Models

Along with our own trained models, we have also evaluated models trained by [27]. These models have an encoder-decoder architecture and are trained on A* generated traces on 30x30 mazes [4]. The mazes are generated by their random generation method as described in Section 3. We see that across model sizes (from 15M to 175M parameters) there are a significant number of instances where the model produces a correct plan but the trace that it outputs is invalid. This is in line with the results of our models and provide further evidence that trace accuracy is not a perfect predictor of plan accuracy.



15M

	Incorrect Plan	Correct Plan
Invalid Trace	4116	1692
Valid Trace	140	452

45M

	Incorrect Plan	Correct Plan
Invalid Trace	5748	609
Valid Trace	10	33

175M

	Incorrect Plan	Correct Plan
Invalid Trace	4203	1763
Valid Trace	98	335

Figure 4: Plan validity versus trace validity for models trained on correct A* traces on 30x30 mazes, measured across varying model sizes and averaged over five runs (6400 responses per run).

A.3 Training on other maze generation algorithms

We also train models on mazes other than Wilson's. We specifically choose the Searchformer-style mazes as all the Wilson models perform the worst on this data. We generate 50k unique problems and then train on both correct and swapped traces. Similar to what we have seen with the Wilson models, we see that the swapped model performs better than the regular model on two out-of-distribution datasets (DFS and Drunkard's walk).

A.4 Training with a different swap seed

To check if the way the traces have been swapped change any of the already seen trends in performance, we also train on a dataset where the traces were swapped using a different random seed from the previously swapped dataset. As seen in Table 5, we find that the new model still outperforms the regular model on the DFS and Drunkard's walk datasets.

A.5 Solving Sokoban Problems

To test if similar results can be obtained on a different and more complex task with a different transition structure, we repeat our experiments for Sokoban puzzles. Sokoban is a PSPACE-complete

[4] We found that the dataset used to train the models had <1% of instances with incorrect traces. Therefore we created our own A* implementation to ensure complete correctness of the traces within our generated datasets.

Table 4: Performance of Swap and A* trace models across maze distributions. The training data is generated via Searchformer-style trace generation. “Plan Val.” = Plan Validity, “Trace Val.” = Trace Validity within valid plans.

Test Set	Regular A* traces		Swapped A* traces	
	Plan Val.	Trace Val.	Plan Val.	Trace Val.
Wilson	28.5%	81.8%	6.8%	0.0%
Kruskal	31.7%	77.9%	8.6%	0.0%
DFS	12.0%	66.7%	14.3%	0.0%
Drunkard’s Walk	44.3%	87.6%	60.9%	0.0%
Searchformer-style	47.0%	63.4%	23.4%	0.0%

Table 5: Performance of Solution-Only, A* Trace, original Swapped-Trace, new seed Swapped-Trace Models across maze distributions. “Plan Val.” = Plan Validity, “Trace Val.” = Trace Validity within valid plans.

Test Set	Soln. Only	Regular A* traces		Swapped A* traces		Swapped A* traces (new seed)	
	Plan Val.	Plan Val.	Trace Val.	Plan Val.	Trace Val.	Plan Val.	Trace Val.
Wilson	4.0%	50.1%	95.2%	51.6%	0.0%	41.5%	0.0%
Kruskal	2.1%	49.7%	96.2%	51.9%	0.0%	42.2%	0.0%
DFS	2.8%	30.8%	82.1%	41.7%	0.0%	35.9%	0.0%
Drunkard’s Walk	0.0%	2.5%	4.0%	26.0%	0.0%	31.3%	0.0%
Searchformer-style	0.0%	0.0%	0.0%	0.2%	0.0%	0.1%	0.0%

[11], grid-based puzzle where an agent must push each box from its start position onto a designated storage dock. At each timestep, the agent may move one cell up, down, left, or right, and can push—but never pull—exactly one adjacent box into the next cell, provided that both the agent’s target cell and the box’s destination cell are free. We encode the entire level (grid layout, agent and box start positions, and dock positions) as a token sequence similar to that of [27]. The model must output a sequence of valid moves that, when executed, places all boxs on their docks; a plan is correct only if every action is executable in the simulated environment and achieves the goal configuration.

Similar to the A* trace generation described in Section 3.2, we modify the A* implementation to output a linearized execution trace. Whenever the algorithm creates a child node and adds it to the open list, it prints `create worker x y box a b box c d cA cB` and when it closes a node and adds it to the closed list, it prints `close worker x y box a b box c d cA cB`. Here, `x y` denotes the worker location, `a b` and `c d` denote the respective box locations, ‘A’ in cA represents the exact cost from the start state to the node (i.e., the $g(n)$ value) and ‘B’ in cB represents the heuristic estimate from that node to the goal state (i.e., the $h(n)$ value). Similar to Searchformer notation [27], we use the prefix "c" to differentiate between co-ordinates and cost estimations. We compute the heuristic value at each node by finding the sum of the Manhattan distances for every possible pairing of boxes to docks, and then take the minimum over all such assignments as our heuristic value.

Similar to the validator described in Section 4, we construct an A* validator that reverses this process for Sokoban puzzles – it takes in a linearized trace and attempts to simulate the corresponding open and closed list operations to check if they are valid with respect to the semantics of this implementation.

Training and Test Dataset - We use the same problem generation procedure used by [27]. A 7 × 7 grid was sampled and two additional wall cells were added as obstacles to the interior of the map. Then two docks, boxes, and the worker locations were randomly placed. If the sampled task is solvable by A*, then the task was admitted to the dataset. We generate 50,000 Sokoban puzzles to construct our training dataset. We also generate the swap dataset for Sokoban problems.

For the test dataset, we use plan length as a proxy for problem difficulty and generate problems that have a final plan length greater than the mean plan length of training dataset problems.

Table 6: Performance of Swapped, A* Trace, and Solution-Only Models on the Test dataset. "Plan Val." = Plan Validity, "Trace Val." = Trace Validity within Valid Plans

Test Set	Soln. Only	Regular A* traces		Swapped A* traces	
	Plan Val.	Plan Val.	Trace Val.	Plan Val.	Trace Val.
Sokoban	18.1%	1.1%	0%	2.3%	0%

Even in the case of Sokoban problems, we see that the correct traces do not help the model perform better than swapped (and incorrect) traces (as shown in Table 6). This reinforces our point that trace accuracy and plan accuracy are not semantically co-related.