EXCESSIVE SUPERVISION AND SHORTCUTS PREVENT IN-DOMAIN LEARNING OF TRIVIAL GRAPH SEARCH

Arvid Frydenlund *
University of Toronto, Computer Science
Vector Institute
arvie@cs.toronto.edu

ABSTRACT

This work concerns the path-star task, a minimal example of searching over a graph. The graph, G, is star-shaped with D arms radiating from a start node, s. A language model (LM) is given G, s, and a target node, t, which ends one of the arms and is tasked with generating the arm containing t. The minimal nature of this task means only a single choice needs to be made: which of the D arms contains t?

Decoder-only LMs fail to solve this simple task above $1/D$ chance due to a learned shortcut that absorbs training supervision. We show how this pathology is caused by excess supervision and present a series of solutions demonstrating that the task is solvable via decoder-only LMs. We find that the task's minimal nature causes its difficulty, as it prevents task decomposition. Our solutions provide insight into the pathology and its implications for LMs trained via next-token prediction.

1 INTRODUCTION

The path-star task is a seemingly simple minimal graph search task intended to exhibit a flaw in the standard next-token prediction paradigm used to train decoder-only autoregressive LMs via teacher-forcing (TF) (Bachmann & Nagarajan, 2024).

Each graph is star-shaped with D arms rooted at a single start node, s. The LM is given the complete graph (as a shuffled edge list) and a query, (s, t), where t is a target node that ends an arm. The task is to generate the arm with t from s to t (Fig 1). This requires the LM to choose an arm by initially generating one of the D *leading* nodes adjacent to s with the rest of the arm being dictated by following edges. Thus the difficulty lies in choosing the correct leading node, l_t, which necessitates planing and reconstruction of the correct arm from t to l_t.

Training via TF conditions the LM on prior ground-truth tokens. This induces learning an undesired shortcut, the *Clever Hans Cheat* (CHC), which allows for trivial prediction of all non-leading nodes via a single edge look-up given the preceding node (given via TF). Thus, all the sequential supervision is absorbed into learning the CHC except a single target token, l_t, which becomes the sole support for learning the required arm reconstruction subtask. As a result, LMs fail to generate the correct arm above the random baseline of $1/D$ chance (Bachmann & Nagarajan, 2024). While it has been shown that decoder-only LMs can express the task (Frydenlund, 2024), **it remains an open question if decoder-only language models trained via teacher-forcing can learn the task**.

1.1 SIGNIFICANCE OF THE FAILURE

LMs are the ubiquitous model for NLP tasks (Brown et al., 2020), as well as for reasoning tasks (Bubeck et al., 2023). These tasks often require planning, which LMs struggle with (Valmeekam et al., 2023; Kambhampati et al., 2024). Potentially, this poor planning performance may be attributable to a fundamental problem with the next-token prediction paradigm. **The path-star task is designed to support such a claim, where the minimal nature of the planning task is mean to isolate and highlight the failure; if standard LMs trained in standard ways fail to solve such a brutally simple task, it calls into question the sufficiency of the standard paradigm.**

*This is a non-archival workshop version of this work. See Frydenlund (2025) for the full version.

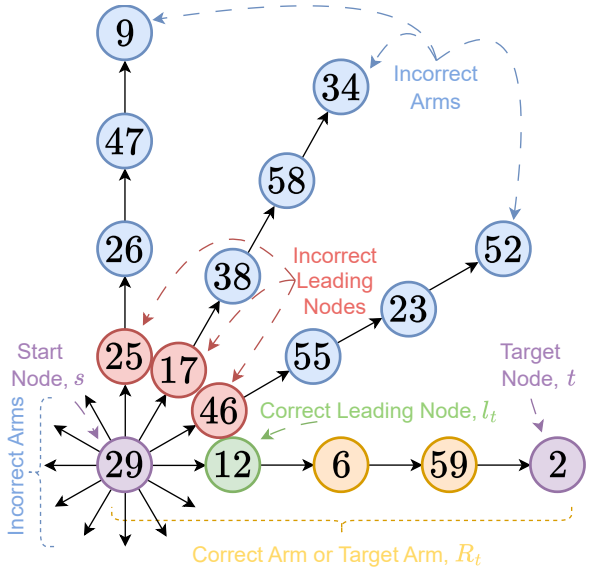


Figure 1: An example path-star graph. $D = 12$, $M = 5$, s is '29', t is '2', R_t is '29 12 6 59 2', and l_t is '12'. We omit eight incorrect arms for space. **The task is to generate R_t given a query, $Q = (s, t)$, and the graph, G, as a tokenized shuffled edge list** (See Fig. 2).

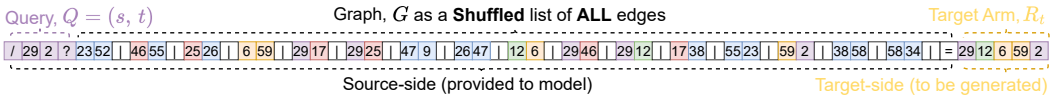


Figure 2: *A tokenization* corresponding to Fig. 1.

This motivated the use of alternative models. Bachmann & Nagarajan (2024) used a 'teacher-less' model that foregoes TF by conditioning on fully masked input (Monea et al., 2023). Frydenlund (2024) generalized this to non- and iterative-autoregressive models and demonstrated learnability differences between types of models. Saparov et al. (2025) also showed positive results on path-star graphs with encoder-only models and on more general graph topologies with both encoder- and decoder-only models.[1] They found that the topology is critical to generalization but that learning does not scale with graph size (and using scratchpads do not solve this issue), leading to the claim that 'transformers struggle to learn to search'. Yin et al. (2024) and Hu et al. (2024) both proposed novel model architectures on the perceived deficiency of decoder-only models in solving the path-star task.

2 Task, Data, and Tokenization

Each graph, G, has D arms of the same length M (inclusive of s) and is constructed by sampling nodes from a set of possible nodes, V, without replacement, making the graph size $|G| = D(M-1)+1$. The edges are determined by this sample order. Thus all nodes are unique and *semanticless*.

The task is tokenized as a sequence consisting of the query, $Q = (s, t)$, with start- and end-of-query markers ('/ s t ?'). Each edge, (u, v), is followed by the end-of-edge marker ('u v |'). See Fig. 2. **The entire graph is provided to the language model as a series of edges, which are randomly shuffled**. This destroys any higher-order structural information about the graph, meaning that the task must be solved via following edges. The source-side input into the model is Q followed by G and the end-of-graph marker ('='). We place Q before G due to it being better for decoder-only models (Frydenlund, 2024). Let $R_t = x_1, \ldots, x_M$ be the series of nodes (not edges) from s to t forming the target arm and the sequential target-side supervision. Each experiment uses a model trained from scratch and different-sized graphs are not mixed during training (except in Sec. 3.5).

[1]They did not try path-star graphs with decoder-only LMs.

Frydenlund (2024) identified that the original experimental design leads to spurious correlations and overfitting due to the task's large sample space,

$$Z = \frac{|V|!}{(|V| - D(M-1) - 1)!} \times D. \tag{1}$$

To this end, they proposed using 'structured samples'. While these help, they did not resolve overfitting and increase training time. Instead, we use an online dataset that generates new samples during training. Saparov et al. (2025) also used an online dataset. Sanford et al. (2024) found better learnability with online training for the k-hop task. We also minimize the space by using $|V| = |G|$.

Information can only be routed into the future due to the decoder's causal constraint. This increases the task's difficulty as the LM must learn two separate routing rules subject to edge $(u,\, v)$ proceeding or succeeding $(v,\, w)$. To avoid this, Frydenlund (2024) introduced an 'arm-wise shuffle' which only shuffles arms relative to each other. However, this allows for a trivial shortcut solution. Instead, we present a 'causal-wise shuffle' where each arm is in sequential order but not contiguous. This alternative setup acts as a control to indicate if the causal constraint is causing difficulties.

2.1 Supervision Adulteration

Above, we discussed how the models will overfit due to spurious correlations in the data. The CHC is also a shortcut learnt due to overfitting, however, it is a different kind of overfitting as it is not caused by the data, but rather by the way the task is constructed. In particular, the CHC is a shortcut caused by providing excess supervision or *adulteration*.

Consider the various ways the task is supervised. In a supervised learning framework, we generally think of 'the supervision' being the target labels as they can be human-annotated. With the next-token prediction paradigm we forego human-annotation by employing a self-supervised rule for generating the targets. In both cases, however, the targets are a function of the input and thus the choice of input is just as much a form of supervision as the targets. Under this view, there are three types of supervision given to the model during training: the target-side labels, target-side input, and source-side inputs.

The path-star task (PST) is designed to induce a bad interaction between these different types of supervision under standard training. For a given step i, the LM is trained to predict the target-side label $x_i \in R_t$. However, it will be given $x{<}\, i$, including x_{i-1} as target-side input due to TF. This induces the trivial single-edge lookup as the edge $(x_{i-1},\, x_i)$ is provided in the source-side input.

Thus excessive supervision partitions the sequential target-label supervision into supporting two tasks: the desired PST, supported by a single target label, and the undesired single-edge lookup task, supported by the remaining labels. This indicates the task is not constructed properly. We will show that 1) this bad interaction, and thus the CHC, can be avoided in various ways and 2) avoiding the CHC is not actually critical for learning the task, if we consider other ways the task is supervised.

In general, how the task is constructed is a form of supervision which encompasses multiple design decisions. We considered that target-side above, however, the source-side representation is also supervised. For example, how the G is shuffled matters (edge-wide vs. casual-wise Tbl. 1). As well as the decision to place Q after or before G or which tokens to include in the query (Sec. 3.3).

There are also non-representational forms of supervision in creating the training data and training procedure. Bachmann & Nagarajan (2024) consider training and evaluating each model of graphs of the exact same size. This is done to keep the task in-domain so as to explicitly rule out out-of-domain effects. Alternatively, we can train the models on various sizes (Sec. 3.5). To elucidate why this is supervision with an example, we could also employ curriculum learning, which would supervise the order of training data to guide training from easy to hard (Bengio et al., 2009).

Our choices of supervision to include s and t in Q and only considering same-sized graphs leads to learning shortcuts for trivially predicting s and t. These are not bigram-based like the CHC, but positional as they are given in the source-side and always appear in the same place in the target-side.

3 METHODS AND EXPERIMENTS

We use decoder-only models with 2 heads, 64 dim. embeddings, 256 dim. feed-forward layers, and learned positional embeddings. We use $L = 8$ layers for all experiments. We use a learning-rate of $5 * 10^{-4}$, a batch-size of 1024, and no dropout. We train for 100M samples.

We modify the task setting from Bachmann & Nagarajan (2024) by, a) placing Q before G, b) using an online dataset to avoid overfitting, and c) setting $|V| = |G|$ instead of 100. **These changes are immaterial to any conjecture regarding an inability to plan and do not prevent learning the CHC.**. We confirm the PST is still unlearnable in this setting in Tbl. 1 in Appx. C.1.[2] Even with very small graphs $|G| = 9$ nodes. This also shows the PST learnable with the causal-wise shuffle, **indicating that the causal constraint accounts for some of the task's difficulty.**

3.1 TOKEN MASKING

We first consider the use of token masking to address the adulteration. This will discourage learning the CHC by preventing conditioning on prior ground-truths during training thus breaking the interaction by modifying the target-side inputs. This is motivated by the limited successes of the 'teacher-less' and iterative- and non-autoregressive models (Frydenlund, 2024). Our simple innovation from these is that we do not need to do full masking, unlike the former, and we can keep the causal parameterization of the model, unlike the latter.

This can be achieved via token dropout/masking (Gal & Ghahramani, 2016) or scheduled sampling to create token replacements (Bengio et al., 2015).[3] The latter has the benefit of providing an anti-CHC learning signal as the model can not trust edge look-ups, but it also introduces more complex noise. Importantly, both are ubiquitous data-noising methods used with standard TFed training. We use contiguous span sampling instead of uniform sampling (Joshi et al., 2020).

Tbl. 3 in Appx. C.2) shows that masking makes the task learnable but struggles as D and M scale. We find minor differences in the two masking types and try mixing them, as they may provide different benefits (token replacement tells the model not to trust single-edge lookups while a masked token prevents these). Finding that a given method makes the PST learnable but does not scale will be a consistent pattern across methods. **Our focus for these methods is a) showing that the task becomes learnable and b) explaining why.** We conjecture about scalability limitations in Appx. B.

We show how masking prevents the CHC in Fig. 3. First, consider the CHC in Fig. 3a and the needed alg. for predicting l_t in Fig. 3b. The CHC learns a forward alg. from the prior token, while the needed alg. must work backward from the target query. Figs. 3c and 3d show how masking induces multi-edge lookups. In Fig. 3c, when all prior tokens are masked, it induces learning a subset of steps for the needed alg. **This provides a deeper explanation for why masking works beyond preventing the CHC; it induces task decomposition** and explains why unadulterated sequential supervision is important. Decomposition occurs because reconstruction is inherently recursive. Fig. 3d shows that having unmasked prior tokens may lead to learning the forward alg. While these may seem similar to a human, they are different algs. and it's unclear if they mutually support learning.

3.2 SCRATCHPADS (SP) TO INCREASE SUPERVISION

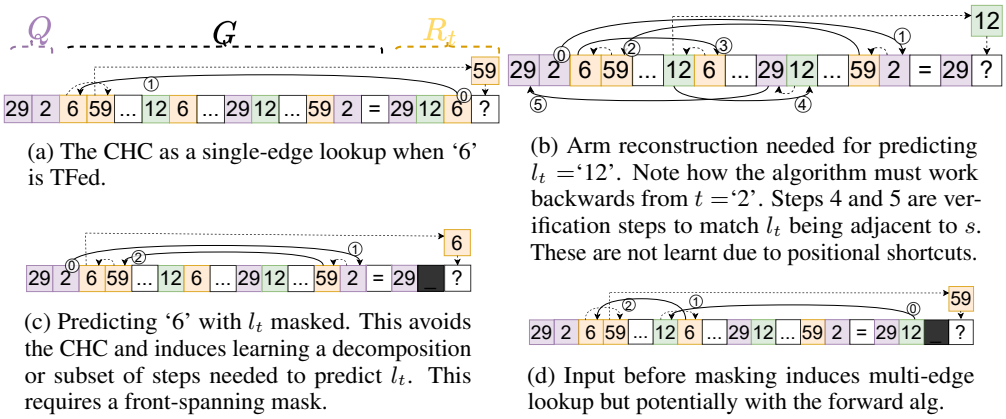
SPs predict an intermediate sequence before the target sequence, providing auxiliary input and target supervision (Nye et al.). Both the reverse arm order and the arm-wise graph order make the task trivial and so would be obvious SPs. However, they are problem-specific and do not prevent adulteration.

Instead, for arm-reconstruction (AR-SP), we generalize the reverse order to generate the arm nodes as a BOW in any order. As there are $M!$ orderings, we use LS over the choices. This unifies the BOW auxiliary and next-token prediction distributions as the next M tokens are the BOW. We can avoid LS via a canonical ordering by sorting by node values. **This introduces node semantics**.

We also use SPs that reconstruct the entire graph by ordering the arms (GR-SP). Full reconstruction would cause adulteration, so we just match leading- and target-node pairs. We order the arms by leading- or target-node value, again, introducing semantics (Fig. 10 in Appx. C.5).

[2] All tables are in appendices due to space constraints.

[3] As we know how the model generates, we skip real scheduled sampling and just sample from V instead.



(a) The CHC as a single-edge lookup when '6' is TFed.

(b) Arm reconstruction needed for predicting l_t ='12'. Note how the algorithm must work backwards from t ='2'. Steps 4 and 5 are verification steps to match l_t being adjacent to s. These are not learnt due to positional shortcuts.

(c) Predicting '6' with l_t masked. This avoids the CHC and induces learning a decomposition or subset of steps needed to predict l_t. This requires a front-spanning mask.

(d) Input before masking induces multi-edge lookup but potentially with the forward alg.

Figure 3: Algorithmic steps performed in the CHC and arm reconstruction, also with masking.

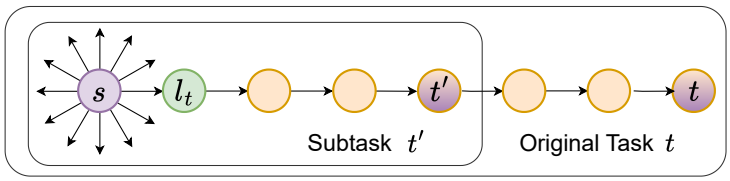


Figure 4: Sampling t as task decomposition.

AR-SP results can be seen in Tbl. 5 in Appx. C.4. The reverse SP is trivially learnt as expected. The BoW SP starts failing to find the solution when $M = 7$ and the ordered SP when $M = 9$. While these make the PST learnable (on small scales), their performance is disappointing. However, their failures are informative. Note that the models fail due to incorrectly predicting the SP and thus condition on incorrect info. The BoW performance is informative as the obvious solution (to a human) would be to generate the BoW in the reverse order. Not learning this shows **the reverse solution is only trivial when it is provided with direct supervision (the same supervision which causes the CHC)**.

The performance of the sorted SP is harder to explain. Sorting naturally decomposes, but, by design, is agnostic to graph edges (expect for the identification step). It may be that this subtask does not mutually support learning the PST task. However, the issue is that the model fails to learn to sort at all at scale, so we suspect that this is the same scaling issue affecting the other methods.

GR-SP results can be seen in Tbl. 6 in Appx. C.5. These fail to learn to solve the task in 76/80 runs. This is exceedingly informative. We have four variants of the GR-SP, where we can go from leading- to target-nodes or vice versa and then we can either sort by leading- or target-node values. Fig. 11 plots the accuracy of each SP token across training. The models learn to correctly identify the needed sets of leading at target nodes. This is done by single-edge shortcuts; leading nodes by adjacency to s and targets nodes by having no following edge. The models also correctly learn to sort either the leading- or target-nodes. This means that in the SP where we go from leading-to-targets, sorted by leading nodes the model can correctly identify the first leading-node but fails to connect it with its paired target. The same thing happens using targets-to-leading, sorted by target nodes. In both cases, **the model knows which arm to reconstruct and but does not learn to do so.**

Here all the model needs to do is deterministic path-following with no planning to choose the correct arm. **This begs the question: if the solution is deterministic and does not require choices, is this actually a planning problem?** These and the BoW SP results indicate that arm-reconstruction is what makes the task hard. As we do not do the full graph reconstruction, we do not provide supervision for path-following or task decomposition. Thus these negative results are consistent with and indirectly support the theory that supervised task decomposition is necessary.

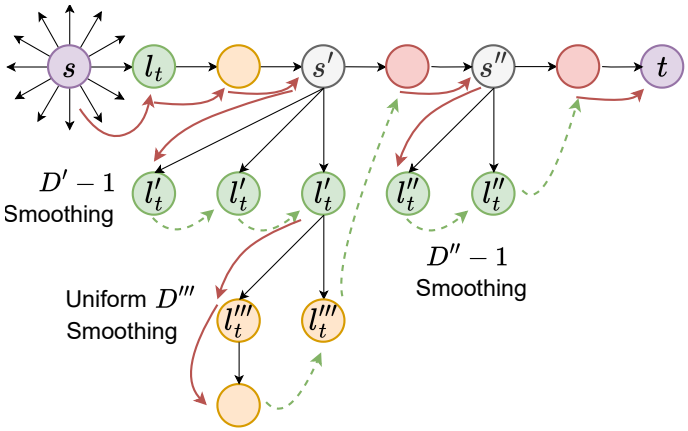


Figure 5: Red and green-dashed arrows form the desired pre-order traversal; red edges are included in G, while green are not, and hence are immune to the CHC.

3.3 Generalized Queries

Given the sensitivity conjecture (Appx. A.1), we consider if providing more than one node from R_t in Q will decrease sensitivity. We do this by sampling a subset of R_t (in any order to avoid adulteration). This is similar to token masking except on the source-side. During inference, only t is given.

Tbl. 7 in Appx. C.6 shows that using a subset of R_t makes the task learnable for small graphs. To verify that that is due to decreasing sensitivity, we tried sampling a general single node from R_t to use as the target in Q. We find that not only does this make the task learnable, it performs better than using a subset. We expect this is because a subset introduces too much noise. However, if this does not work because of reduced sensitivity, why does it work? Because it induces task decomposition (illustrated in Fig. 4). Hu et al. (2024) performed this same experiment and found it did not learn the task. We find that this version of the task is still hard to learn in the original setting.

3.4 From Path-Star to Tree-Star

One way of preventing the CHC is to remove or modify edges in G to prevent single-edge lookups. We achieve this by generalizing the task to consider arms as trees instead of paths. In particular, we are going to train on trees but evaluate on path-star graphs. Training on trees slightly changes the training objective as we are no longer generating the arm but an equivalent pre-order traversal of the tree. This introduces a problem as a traversal requires a planner tree to determine the order of multiple child nodes in the traversal. Encoding it as a planner tree will induce new undesired shortcuts.

We employ a trick to avoid this. By task definition, t must be the last generated token. This precludes any sub-tree containing t from being generated before the others. This makes the distribution over D' child nodes asymmetric but uniform over $D' - 1$ choices. However, this only works for sub-trees containing t. This is achieved via label smoothing over valid child nodes during training. Call these *D-ary trees*. See Fig. 5. Following this logic, we can design the tree to resolve any ordering ambiguity with a deterministic traversal by only only allowing sub-trees containing t to have a maximum of 2 children and all others one child. Call these *split trees*. See Fig. 6.

Tbl. 8 in Appx. C.7 shows that training on split trees makes the task learnable. This can again be explained as inducing task decomposition, where each split creates a $D' = 2$ path-star subtask (except one where both arms are generated). It is interesting that $D > 2$ works, as the induced subtask is restricted to $D' = 2$. However, we find that the D-ary trees do not generalize. We conjecture that smoothing over sub-trees with D choices creates a deficient subtask that mirrors the adulterated PST.

Training on trees and evaluating path-star graphs may look like an exotic solution, but we stress this is actually a generalization of graph topology and one that does not go far enough. We suspect that the best graph topology would be one which allows for perfect decomposition, where each subtask is exactly the same as the original. This explains why one needs to carefully when choosing graph

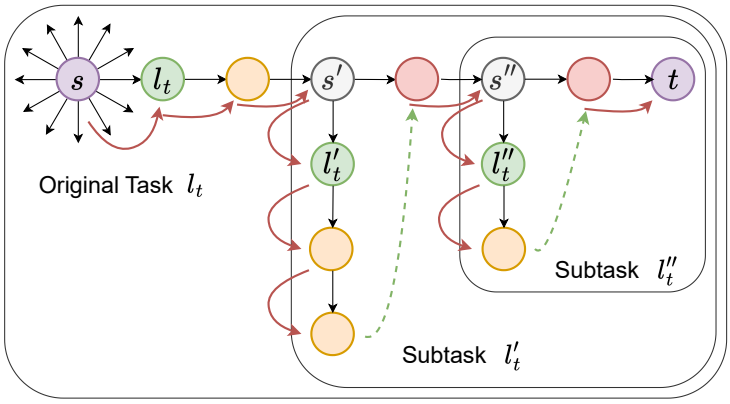


Figure 6: A split tree. Each split induces a decomposition and there is only a single valid pre-order traversal. Not the subtask does not exactly mirror the original.

topology (Saparov et al., 2025). Training on trees and evaluating on path-star graphs is also interesting in that it is counterintuitive from the perspective of in-domain learning – we require training on trees to generalize to paths when training on paths directly fails.

3.5 Generalized Length Decomposition

If we want to induce task decomposition, an obvious way would be to supervise the training process by sampling different-sized graphs. Tbl. 9 in Appx. C.8 shows this makes the task learnable as expected. Training on various values of D does not seem to help, but combining general length and target sampling does. We expect these results would be better if given more training time, as this introduces a lot of noise. These results show that training on various lengths is not just for out-of-domain generalization but also promotes in-domain learning. This also prevents positional shortcuts.

4 Conclusion

We have taken the PST from being unlearnable to learnable, shown how the original task is designed with adulterated supervision and explained why this makes it unlearnable due to not providing any decomposition supervision, and developed methods which retain standard training to overcome this.

We have empirically demonstrated that decomposition is critical for learning to search over graphs. Our work serves as a bridge between Bachmann & Nagarajan (2024) and Saparov et al. (2025) by providing explanation for why the graph search task presented by the former is seemingly unlearnable, while the very similar graph search task presented by the latter is learnable; the latter provides decomposition supervision, while the former does not due to adulteration. Specifically, Saparov et al. (2025)'s task setup incorporates general queries, lengths, and graph topologies. Note, these methods still permit the CHC. We suggest that masking will be important for scaling search to larger graphs.

Our results suggest that the core difficulty of the task does not concern planning but graph reconstruction. They also show that seemingly trivial solutions will not be found unless directly supervised. If one was concerned about the implications that the empirical results of the path-star task had on the sufficiency of the next-token prediction paradigm for planning task, this work alleviates those concerns. If one is skeptical about such conjectures, this validates their beliefs with explanation.

Acknowledgments

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

REFERENCES

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2296–2318. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bachmann24a.html.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1171–1179, Cambridge, MA, USA, 2015. MIT Press.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. Simplicity bias in transformers and their ability to learn sparse Boolean functions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5767–5791, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.317. URL https://aclanthology.org/2023.acl-long.317.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Sébastien Bubeck, Varun Chandrasekaran, 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. *arXiv preprint arXiv:2303.12712*, 2023.

Yingshan Chang and Yonatan Bisk. Language models need inductive biases to count inductively. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s3IBHTTDYl.

Arvid Frydenlund. The mystery of the pathological path-star task for language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 12493–12516, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.695. URL https://aclanthology.org/2024.emnlp-main.695.

Arvid Frydenlund. Language models, graph searching, and supervision adulteration: When more supervision is less and how to make more more. *arXiv preprint arXiv:2503.10542*, 2025.

Arvid Frydenlund, Gagandeep Singh, and Frank Rudzicz. Language modelling via learning to rank. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10):10636–10644, Jun. 2022. doi: 10.1609/aaai.v36i10.21308. URL https://ojs.aaai.org/index.php/AAAI/article/view/21308.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/076a0c97d09cf1a0ec3e19c7f2529f2b-Paper.pdf.

Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14973–15008, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.800. URL https://aclanthology.org/2024.acl-long.800.

Edward S Hu, Kwangjun Ahn, Qinghua Liu, Haoran Xu, Manan Tomar, Ada Langford, Dinesh Jayaraman, Alex Lamb, and John Langford. Learning to achieve goals with belief state transformers. *arXiv preprint arXiv:2410.23506*, 2024.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL https://aclanthology.org/2020.tacl-1.5.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=Th8JPEmH4z.

Giovanni Monea, Armand Joulin, and Edouard Grave. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

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. In *Deep Learning for Code Workshop*.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43276–43327. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sanford24a.html.

Abulhair Saparov, Srushti Ajay Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Mehran Kazemi, Najoung Kim, and He He. Transformers struggle to learn to search without in-context exploration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9cQB1Hwrtw.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=X6dEqXIsEW.

Yongjing Yin, Junran Ding, Kai Song, and Yue Zhang. Semformer: Transformer language models with semantic planning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18669–18680, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1039. URL https://aclanthology.org/2024.emnlp-main.1039.

A Appendix

A.1 Sensitivity Conjecture

Why learning l_t from a single target token is difficult is an open question. Frydenlund (2024) conjectured it relates to the task being sensitive to a single token, t. Hu et al. (2024) provided a construction of parity as a path-star task that only generates l_t, implying it is at least as hard to solve as parity. Parity is maximally sensitive and known to be extremely difficult to learn with transformers (Bhattamishra et al., 2023; Hahn & Rofin, 2024). This conjecture motivates some of our methods, however, we find little empirical support for it (Sec. 3.3).

B LIMITATIONS

Scaling Issues: The major limitation of our work is that each method fails to scale with either D or M. We believe that using graph topologies that allow for stronger and more consistent decomposition where each subtask mirrors the main task will be key for scalability. Thus we think that using path-star graphs will not be the best environment to consider how the methods we have developed scale to larger graphs. Instead, we believe using balanced graphs would be a better environment (Saparov et al., 2025). This is beyond the scope of this work and we leave it for future work.

We also believe that there may be different issues affecting the scaling of D and M. We know that M scales with the number of model layers (Frydenlund, 2024). It is unclear if learning the logarithmic algorithm is harder than the linear algorithm. As we use a model with 8 layers and often find scaling past $M = 9$ difficulty, this may account for some scaling issues for M. We have tied the size of the vocabulary to D by setting $|V| = |G|$. This may account for some scaling issues for D.

Ranking-into-the-Future: Because using path-star graphs is not a good environment to test the scalability of our methods, we have introduced ranking-into-the-future but do not believe we have evaluated its full potential for planning tasks. Our discussion around RIFT is framed as being a change in distribution from next-token prediction. While our goal of this work is to show that next-token prediction is sufficient for learning the task, we do not consider what is the best method.

Alternative examples of adulteration: We describe adulteration as a generic issue but only talk about it in the context of graph searching. However, we believe it is a useful term for identifying similar issues and showing they can be solved via similar methods. For example, Chang & Bisk (2025) consider trying to learn to count by providing a model with a contiguous sequence of numbers and training using next-token prediction. However, as they point out, this will be ineffective due to trivial bigram shortcuts. This is because they have presented the task in an adulterated form.

Why is task decomposition necessary: We have empirically shown that task decomposition is necessary for learning the task, however, this does not explain why it is necessary. That is, characterizing the core underlying difficulty is still an open question. We expect solving this will be insightful for learnability theory of transformers.

While we did not find positive results trying to directly decrease sensitivity by using multiple query nodes, our results are indecisive and may be explained by an increase noise. Thus the sensitivity conjecture is still a possible explanation.

C EXPERIMENTS

For each experiment, we report the results of $n = 5$ differently seeded runs (with the exception of a few experiments where runs failed due to computational issues). We find a high variance for the number of iterations needed to solve the task between runs. This makes considering multiple runs important when considering if a given experiment is learnable or not. Note when we say 'unlearnable', this does not mean the task is provably unlearnable but is rather shorthand for 'not found to be empirically learnt given 100 epochs'.

We abuse the term 'epoch' to mean 1M samples and do so for reporting results. This is because there are no true epochs when using online datasets.

C.1 BASELINES

Tbl. 1 provides baseline results of the path-star task (PST) using both edge- and causal-wise shuffling of G. We find that the PST is unlearnable, even when using online training and reducing the sample space by setting the vocabulary size to the graph size ($|V| = |G|$) and tokenizing Q After G (Frydenlund, 2024). This is consistent with the results of Bachmann & Nagarajan (2024). Tabl. 2 provides a more fine-grained breakdown of the accuracy of the positional accuracy for l_t and the following to nodes for each run. This shows that l_t is predicted at random chance of $1/D$ while the other two nodes are predicted with 100% accuracy due to the CHC. This behaviour is illustrated in Fig. 7. Note that the start and target nodes are learnt immediately due to positional shortcuts (not the CHC). One exception is Run 3 of the exp. where $D = 5,\ M = 7$ which fails to learn the CHC (Fig. 8). This demonstrates the CHC is not guaranteed to be learnt, despite its seeming simplicity.

We find that using a causal-wise shuffle makes the PST learnable on small graphs. This indicates that the causal constraint accounts for some of the task's difficulty. However, once we consider arms with moderate length ($M = 9$), the task is no longer perfectly learnt (at least within the 100 epochs provided). Fig. 9 shows l_t being fitted by in the causal-wise version of the task. This also shows that, even when the task is learnt, the CHC is still employed when solving the task, otherwise we would expect a change in overall accuracy of the other nodes as they become predicted by the new algorithm.

				Test-Force R_t		Test-Gen R_t	
ID	Desc.	D	M	SR	ABB	SR	ABB
	Edge-Wise	2	5	0%	0%	0%	0%
		3	5	0%	0%	0%	0%
		4	5	0%	0%	0%	0%
		5	5	0%	0%	0%	0%
	$\|V\| = \|G\|$, Online Training, Q After G	2	7	0%	0%	0%	0%
		3	7	0%	0%	0%	0%
		4	7	0%	0%	0%	0%
		5	7	0%	0%	0%	0%
	Causal-Wise	2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	100%	100%	100%	100%
		5	5	60%	100%	60%	100%
		2	7	100%	100%	100%	100%
		3	7	60%	80%	60%	80%
		4	7	0%	20%	0%	20%
		5	7	0%	40%	0%	40%
		2	9	40%	100%	40%	100%
		3	9	0%	40%	0%	40%
		2	12	0%	80%	0%	80%
		3	12	0%	20%	0%	20%

Table 1: Baseline results. We report the *Success Rate* (SR) where the model predicts $> 95\%$ sequential accuracy and *Above-Baseline* (ABB) where the model predicts $> (100/D+10)\%$ sequential accuracy. When this happens it indicates that the model can predict l_t in some cases. As such, when ABB $>$ SR, it implies that the model would have learnt the task had it been provided with more training time in these cases. We report on the test partition using both 'teacher-forced inference' which conditions on the correct sequence regardless of past inaccuracies (Test-Force R_t) as well as true auto-regressive generation (Test-Gen R_t). In general, these provide the exact same results, since in both cases, l_t will either be learnt at $> 95\%$ accuracy or not, leading to the same overall sequential accuracy. Results are reported after 100 epochs i.e. 100M training samples.

		Run 1			Run 2			Run 3			Run 4			Run 5		
D	M	l_t	2	3	l_t	2	3	l_t	2	3	l_t	2	3	l_t	2	3
2	5	50%	✓	✓	50%	✓	✓	50%	✓	✓	50%	✓	✓	50%	✓	✓
3	5	33%	✓	✓	33%	✓	✓	33%	✓	✓	33%	✓	✓	33%	✓	✓
4	5	25%	✓	✓	25%	✓	✓	25%	✓	✓	25%	✓	✓	25%	✓	✓
5	5	20%	✓	✓	20%	✓	✓	20%	✓	✓	20%	✓	✓	20%	✓	✓
2	7	50%	✓	✓	50%	✓	✓	50%	✓	✓	50%	✓	✓	50%	✓	✓
3	7	33%	✓	✓	33%	✓	✓	33%	✓	✓	33%	✓	✓	33%	✓	✓
4	7	25%	✓	✓	25%	✓	✓	25%	✓	✓	25%	✓	✓	25%	✓	✓
5	7	20%	✓	✓	20%	✓	✓	4%	4%	4%	20%	✓	✓	20%	✓	✓

Table 2: Training positional accuracy for l_t, pos. 2, and pos. 3 for the edge-wise baseline results in Tbl. 1. ✓ indicates 100% accuracy. In all but a single run, the CHC is learnt for pos. 2 and 3 (and all other non-leading nodes) and l_t is predicted at $1/D$ chance. The single run that fails to learn the CHC corresponds with the failure in Fig. 8.

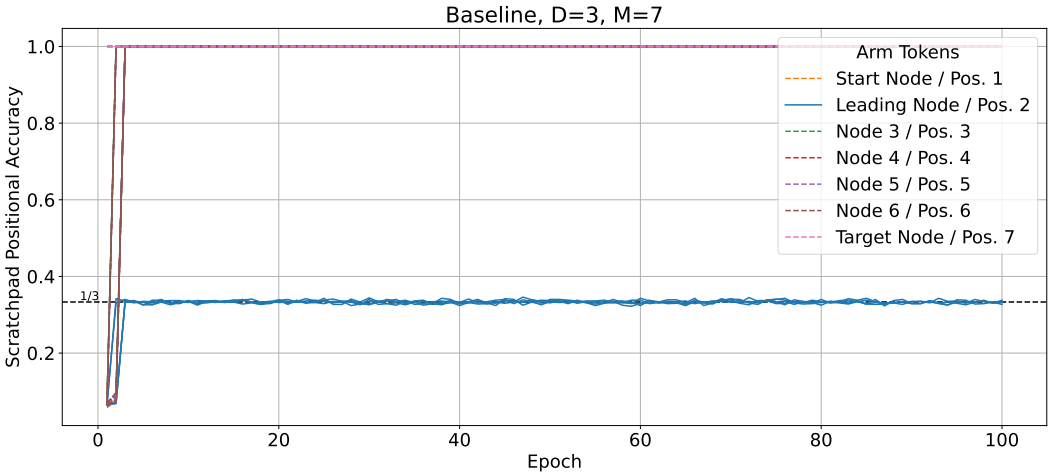


Figure 7: A baseline demonstrating multiple shortcuts used to learn all nodes except the leading node. The start and target nodes can be immediately learnt by positional shortcuts, while nodes 3-6 are learnt by the bigram CHC. The leading node is only predicted at chance accuracy of $1/D$. These consider 'teacher-forced' inference which conditions on the correct sequence regardless of past inaccuracies. We use online training so each 'epoch' is 1M sampled examples. It is over five seeded runs.

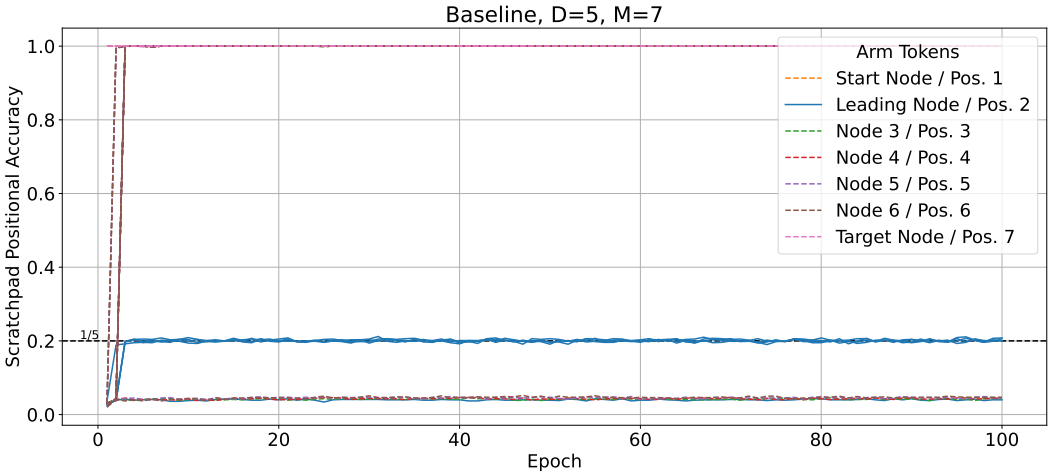


Figure 8: A baseline where the CHC was not learnt for all non-leading tokens.

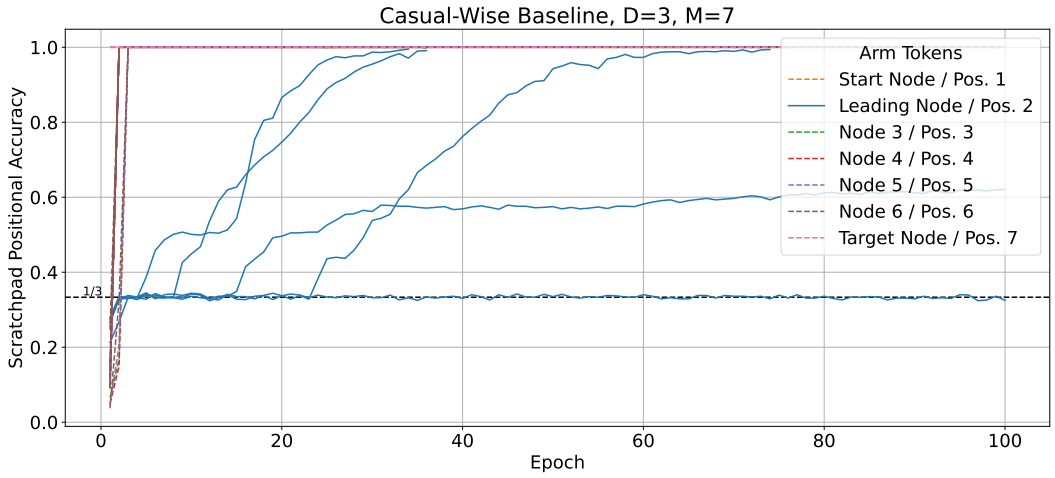


Figure 9: Using the causal-wise ordering of edges allows the task to be learnt on 3/5 runs with one run expected to be learnt.

Because the PST in its original form is unlearnable, it is impossible to hyper-parameter tune it. As such, we also use the causal-wise version to determine valid hyper-parameters for all experiments. The reported results, for the baselines and all other experiments, are after we have found valid hyper-parameters using the causal-wise version of the task and are thus consistent for all experiments.

C.2 Masking Results

Tbl. 3 provides results using span masking via token dropout, token replacement, and a mixture of both.

C.3 Alternative Sequential Distributions

We consider changing the learnt distribution from one over the next token to one over the next tokens. This is done via learning a belief-state, B, which is a hidden-state that supports making future predictions via some linear function of B i.e., $P_B(x_{i:M} \mid f(B = x_{<i}))$ (Hu et al., 2024).

We present three simple methods for learning this future distribution, P_B: bag-of-words (BoW), label-smoothing (LS), and ranking. Yin et al. (2024) used a BoW baseline with R_t as the bag. This performed nearly as well as their proposed model and solved the task in the majority of cases.[4] We exclude prior tokens ($< i$) from the bag so that it only contains future tokens at each step. Note BoW is equivalent to LS with uniform smoothing.

BoW is based on the inductive bias that nodes in R_t are more important than nodes in the other arms. We can extend this with another inductive bias which assumes that near-present tokens are more important than far-future tokens i.e. that the order matters. This can be achieved using monotonically decreasing label weights. Thus LS requires hand-crafted weights to form a hand-crafted distribution that the model tries to match. We can avoid this ad-hoc approach via an equivalence between LS and ranking (Frydenlund et al., 2022).

C.3.1 Ranking-into-the-Future (RITF)

We can go from LS to ranking-into-the-future by constructing rank targets and training with a rank-based loss, providing a structured loss over multiple tokens at each time step. As we are using future tokens, the structure is over the sequence and the sequential order is used for the rank-order.

Let the future distribution *at a single step* i be $P_{B,i}(x_i \succ x_{i+1} \succ \cdots \succ x_M)$ such that the scores of sequential tokens decreases monotonically from time-step i. Let $\sigma_t = f(B = x_{<i})$ be a vector of

[4]Note they used pretrain GPT2 models.

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Gen R_t SR	Test-Gen R_t ABB
	Span Token Dropout	2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	100%	100%	100%	100%
		5	5	60%	80%	60%	80%
		2	7	80%	80%	80%	80%
		3	7	40%	60%	40%	60%
		4	7	20%	20%	20%	20%
		5	7	40%	40%	40%	40%
		2	9	40%	40%	40%	40%
		3	9	0%	0%	0%	0%
		4	9	0%	0%	0%	0%
		5	9	0%	0%	0%	0%
		2	12	0%	0%	0%	0%
	Causal-Wise Token Dropout	5	9	60%	60%	60%	100%
		5	12	0%	66%	0%	66%
	Span Token Replacement	2	5	60%	60%	60%	60%
		3	5	80%	80%	80%	80%
		2	7	100%	100%	100%	100%
		3	7	20%	40%	20%	40%
	Span Mixed Token Dropout and Replacement	2	5	80%	80%	80%	80%
		3	5	80%	80%	80%	80%
		4	5	100%	100%	100%	100%
		5	5	80%	80%	80%	80%
		2	7	100%	100%	100%	100%
		3	7	100%	100%	100%	100%
		4	7	0%	20%	0%	20%
		5	7	0%	20%	0%	20%
		2	9	60%	60%	60%	60%
		3	9	80%	80%	80%	80%
		2	12	20%	20%	20%	20%
		3	12	0%	0%	0%	0%

Table 3: Masking results.

logits or scores. Then we use a pair-wise hinge loss over the entire sequence in R_t s.t.

$$L_B = \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_{k=j+1}^{M} \max(0,\ 1 - (\sigma_{i,j} - \sigma_{i,k})) \tag{2}$$

We incorporate another bias that ranks tokens in R_t above all others, encoding the concept that the correct arm is more important than the others.[5]. This creates very dense supervision with $M(M-1)/2$ intra- and $M^2(|V| - M)$ inner-arm pairs.

C.3.2 Results and Discussion

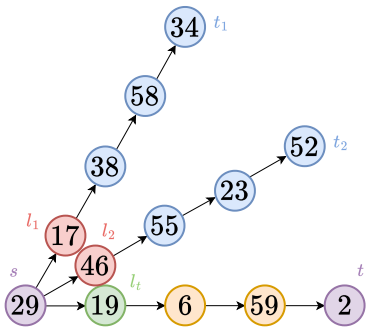
Tbl. 4 shows that RITF is superior to the two other methods. For LS we use a stepped monotonically decreasing weight (Frydenlund et al., 2022). This does not work very well as it couples the inductive bias with loss scaling (so future tokens have tiny weights). We know the inductive bias is not at fault as it is the same as in RIFT. This shows that it is easier to specify rules than specific weights.

Future prediction works for multiple related reasons. First, the loss requires multi-edge lookups and so induces learning arm reconstruction. Second, it avoids adulteration by skipping adjacent inputs for all targets at $> i + 1$. **This is the same as masking, except instead of using noised input to cause the skip, it is implicitly defined as part of the loss.** Third, by applying this at each time-step, **the loss induces task decomposition across the sequence** where learning $P_{B,\,i+1}$ is a sub-problem of learning $P_{B,\,i}$ (and is easier to boot).

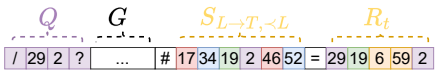
C.4 Arm Reconstruction Scratchpads Results

Tbl. 5 shows the results for arm reconstruction scratchpads. These are the first results that see differences between autoregressive inference and teacher-forced infrequence. Note that sequence accracy is evaluated independently for R_t and the scratchpads. This is why in the teacher-forced setting the model can get 100% SR for R_t but less than that for the SP.

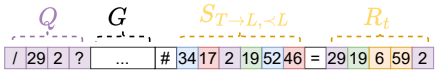
C.5 Graph Reconstruction Scratchpads Results



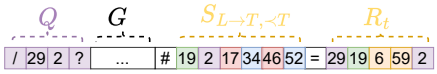
(a) The path-star graph with $D = 3$ and $M = 5$ used when constructing the graph reconstruction scratchpads in Figs 10b, 10d, 10c, and 10e.



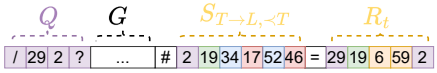
(b) Leading to target pairs, sorted by leading order.



(c) Target to leading pairs, sorted by leading order.



(d) Leading to target pairs, sorted by target order.



(e) Target to leading pairs, sorted by target order.

Figure 10: Illustration of graph reconstruction scratchpads.

Tbl. 6 shows the results for graph reconstruction scratchpads. This shows that only 4 runs succeed in learning the task.

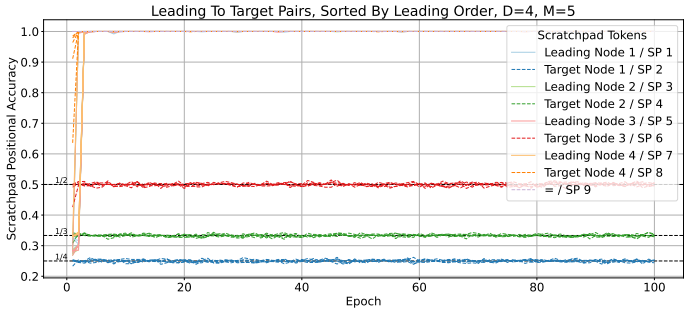
[5]Not in Eq. 2. This is implicit in BOW/LS

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Gen R_t SR	Test-Gen R_t ABB
	Uniform Label Smoothing (BoW)	2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	20%	60%	20%	60%
		5	5	20%	60%	20%	60%
		2	7	60%	60%	60%	60%
		3	7	100%	100%	100%	100%
		4	7	0%	0%	0%	0%
		5	7	0%	0%	0%	0%
		2	9	100%	100%	100%	100%
		3	9	100%	100%	100%	100%
		4	9	20%	60%	20%	60%
		5	9	0%	0%	0%	0%
		2	12	0%	20%	0%	20%
		3	12	0%	0%	0%	0%
		4	12	0%	0%	0%	0%
		5	12	0%	0%	0%	0%
	Monotonically Decreasing Label Smoothing (LS)	3	5	100%	100%	100%	100%
		3	7	100%	100%	100%	100%
		3	9	0%	0%	0%	0%
	Ranking into the Future (RITF)	2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	100%	100%	100%	100%
		5	5	80%	80%	80%	80%
		2	7	100%	100%	100%	100%
		3	7	100%	100%	100%	100%
		4	7	80%	80%	80%	80%
		5	7	60%	60%	60%	60%
		2	9	100%	100%	100%	100%
		3	9	100%	100%	100%	100%
		4	9	100%	100%	100%	100%
		5	9	60%	100%	60%	100%
		2	12	100%	100%	100%	100%
		3	12	100%	100%	100%	100%
		4	12	60%	100%	60%	100%
		5	12	0%	100%	0%	100%
		2	15	60%	100%	60%	100%
		3	15	0%	100%	0%	100%

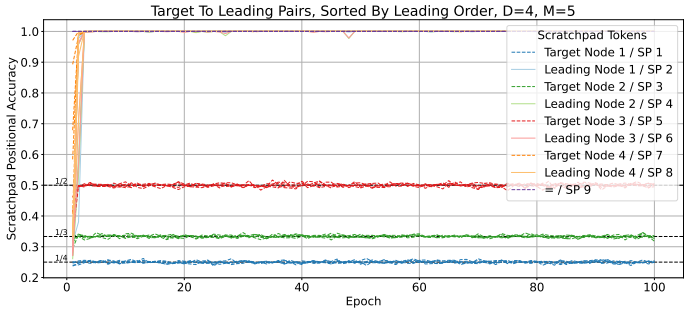
Table 4: Alternative distribution results.

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Force S SR	Test-Force S ABB	Test-Gen R_t SR	Test-Gen R_t ABB	Test-Gen S SR	Test-Gen S ABB
	Reverse	5	5	100%	100%	100%	100%	100%	100%	100%	100%
		5	7	100%	100%	100%	100%	100%	100%	100%	100%
		5	9	100%	100%	100%	100%	100%	100%	100%	100%
		5	12	100%	100%	100%	100%	100%	100%	100%	100%
	BoW	2	5	100%	100%	100%	100%	100%	100%	NA	NA
		3	5	100%	100%	100%	100%	100%	100%	NA	NA
		4	5	100%	100%	100%	100%	100%	100%	NA	NA
		5	5	100%	100%	100%	100%	100%	100%	NA	NA
		2	7	100%	100%	100%	100%	100%	100%	NA	NA
		3	7	100%	100%	100%	100%	100%	100%	NA	NA
		4	7	100%	100%	100%	100%	100%	100%	NA	NA
		5	7	60%	60%	60%	60%	60%	60%	NA	NA
		2	9	100%	100%	100%	100%	100%	100%	NA	NA
		3	9	80%	80%	80%	80%	40%	80%	NA	NA
		4	9	40%	60%	60%	60%	0%	40%	NA	NA
		5	9	40%	40%	40%	40%	0%	40%	NA	NA
	Sorted Arm	2	5	100%	100%	100%	100%	100%	100%	100%	100%
		3	5	100%	100%	100%	100%	100%	100%	100%	100%
		4	5	100%	100%	100%	100%	100%	100%	100%	100%
		5	5	100%	100%	100%	100%	100%	100%	100%	100%
		2	7	100%	100%	100%	100%	100%	100%	100%	100%
		3	7	100%	100%	100%	100%	100%	100%	100%	100%
		4	7	100%	100%	60%	100%	60%	100%	60%	100%
		5	7	100%	100%	40%	100%	40%	100%	40%	100%
		2	9	100%	100%	100%	100%	100%	100%	100%	100%
		3	9	100%	100%	80%	100%	100%	100%	80%	100%
		4	9	100%	100%	0%	100%	0%	100%	0%	100%
		5	9	60%	60%	20%	20%	20%	20%	20%	20%
		2	12	100%	100%	40%	80%	40%	80%	40%	80%
		3	12	100%	100%	0%	40%	0%	40%	0%	40%
		4	12	0%	20%	0%	0%	0%	0%	0%	0%
		5	12	0%	0%	0%	0%	0%	0%	0%	0%
	Forward	2	5	100%	100%	80%	80%	100%	80%	100%	80%
		3	5	100%	100%	100%	100%	100%	100%	100%	100%
		4	5	100%	100%	100%	100%	100%	100%	100%	100%
		5	5	100%	100%	100%	100%	100%	100%	100%	100%
		2	7	100%	100%	75%	75%	75%	75%	75%	75%
		3	7	20%	100%	20%	20%	20%	20%	20%	20%
		4	7	0%	0%	0%	0%	0%	0%	0%	0%
		5	7	0%	0%	0%	0%	0%	0%	0%	0%
		2	9	60%	100%	40%	60%	40%	60%	40%	80%
		3	9	0%	0%	0%	0%	0%	0%	0%	0%
		4	9	0%	0%	0%	0%	0%	0%	0%	0%
		5	9	0%	0%	0%	0%	0%	0%	0%	0%

Table 5: Results for arm-reconstruction scratchpads. We also achieved the same results for the reverse scratchpad for $D \in \{2,\ 3,\ 4\} \times M \in \{5,\ 7,\ 9\}$. For the BOW experiments, there are multiple correct values for each (except for the last) predictive scratchpad step and we did not implement multi-value accuracy for generation (hence 'NA').

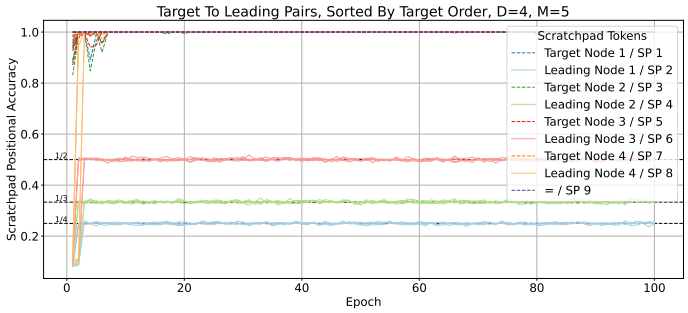


(a) Leading nodes are predictable when sorting by leading order. However, the target leading node can not be predicted even given the correct corresponding leading node. Each plot consists of 5 differently seeded experiments. Note that colours correspond to leading/target index and not scratchpad (SP) index.



(b) Leading nodes are still predictable when sorting by leading order, even when following incorrect target nodes in the scratchpad.

(c) Target nodes are predictable when sorting by target order, even when following incorrect leading nodes in the scratchpad.



(d) Target nodes are still predictable when sorting by target order, however, the correct leading node can not be predicted even given the correct corresponding target node.

Figure 11: Validation set accuracy of the scratch pad tokens across training. These results consider 'teacher-forced' inference which conditions on the correct sequence regardless of past inaccuracies (including the scratchpad).

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Force S SR	Test-Force S ABB	Test-Gen R_t SR	Test-Gen R_t ABB	Test-Gen S SR	Test-Gen S ABB
	$S_{L\to T,\prec L}$	2	5	100%	100%	40%	40%	40%	40%	40%	40%
		3	5	100%	100%	0%	0%	0%	0%	0%	0%
		4	5	100%	100%	0%	0%	0%	0%	0%	0%
		5	5	100%	100%	0%	0%	0%	0%	0%	0%
	$S_{T\to L,\prec L}$	2	5	100%	100%	20%	20%	20%	20%	20%	20%
		3	5	100%	100%	0%	0%	0%	0%	0%	0%
		4	5	100%	100%	0%	0%	0%	0%	0%	0%
		5	5	100%	100%	0%	0%	0%	0%	0%	0%
	$S_{L\to T,\prec T}$	2	5	100%	100%	0%	0%	0%	0%	0%	0%
		3	5	100%	100%	0%	0%	0%	0%	0%	0%
		4	5	100%	100%	0%	0%	0%	0%	0%	0%
		5	5	100%	100%	0%	0%	0%	0%	0%	0%
	$S_{T\to L,\prec T}$	2	5	100%	100%	20%	20%	20%	20%	20%	20%
		3	5	100%	100%	0%	0%	0%	0%	0%	0%
		4	5	100%	100%	0%	0%	0%	0%	0%	0%
		5	5	100%	100%	0%	0%	0%	0%	0%	0%

Table 6: Results for graph-reconstruction scratchpads.

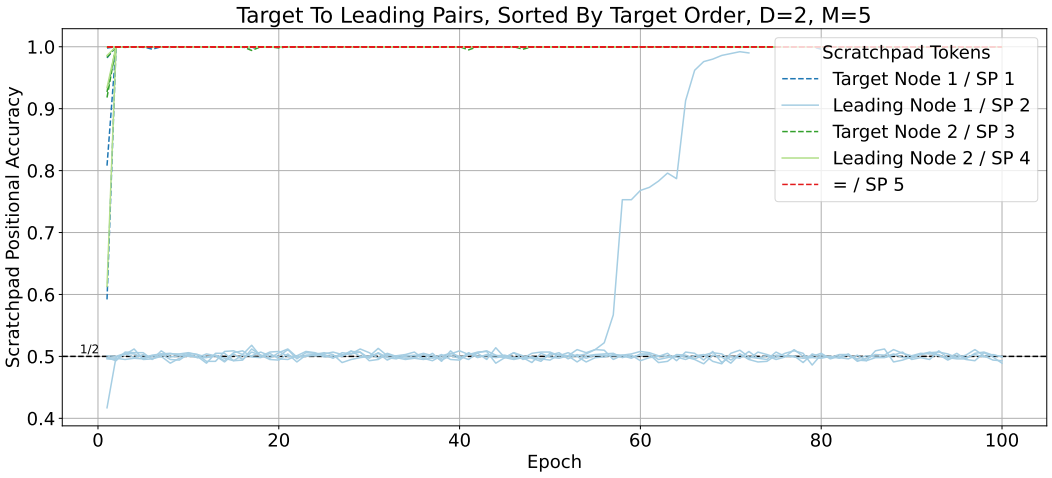


Figure 12: A $D = 2$ graph-reconstruction experiment where one of the runs successfully learnt the task.

Figs. 11a and 11b demonstrate that sorting by leading node leads to all leading nodes being correctly predicted, regardless of whether the leading node precedes or succeeds the corresponding target node. Figs. 11c and 11d show the inverse of this where sorting on the target node leads to all target nodes being correctly predicted. Here we see a consistent pattern where the non-sorted nodes fail to be predicted above chance.

Consider the two cases where the leading node precedes the target node and the arms are sorted by leading node value in Fig. 11a and where the target node precedes the leading node and the arms are sorted by target node value in Fig. 11d. This indicates that the model can correctly predict and thus condition on the correct preceding node but fails to predict the corresponding target or leading node in the next prediction.

C.6 Query Results

Tbl. 7 shows the results of using general queries. When using a single target, only 3/20 experiments succeeded in learning the task with a general target in the original setting compared 16/20 in the

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Gen R_t SR	Test-Gen R_t ABB
	Query Subset (Padded)	2	5	100%	100%	100%	100%
		3	5	75%	75%	75%	75%
		4	5	40%	40%	40%	40%
		5	5	80%	80%	80%	80%
		2	7	60%	100%	60%	100%
		3	7	0%	0%	0%	0%
		4	7	0%	0%	0%	0%
		5	7	0%	0%	0%	0%
		2	9	20%	100%	20%	100%
	General Single Target	2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	80%	80%	80%	80%
		5	5	40%	80%	40%	80%
		2	7	100%	100%	100%	100%
		3	7	80%	100%	80%	100%
		4	7	20%	60%	20%	60%
		5	7	40%	40%	40%	40%
	General Single Target (Original Setting) $\lvert V\rvert$=100, Offline Training, Q Before G	2	5	20%	20%	20%	20%
		3	5	20%	20%	20%	20%
		4	5	20%	60%	20%	60%
		5	5	0%	0%	0%	0%

Table 7: Results for alternative query methods. All results are evaluated with query being only the final node in the arm.

new setting. This indicates that such a finding may be easy to miss. However, as discussed above there is also the issue of hyperparameter-tuning a model which fails to learn a task. One can not hyperparameter-tune models on the unadulterated path-star task as it doesn't learn above chance. Thus another explanation is that, without first finding working models, in our case using the causal-wise shuffling, we may not have properly set the hyperparameters needed for finding these results.

C.7 Tree Results

Tbl. 8 shows the results of the tree experiments.

C.8 Training on Multiple Lengths and/or Degrees Results

Tbl. 9 shows the results for training using a sampled M and/or sampled D.

ID	Desc.	D	M	Test-Force R_t SR	Test-Force R_t ABB	Test-Gen R_t SR	Test-Gen R_t ABB
		2	5	100%	100%	100%	100%
		3	5	60%	100%	60%	100%
		4	5	0%	100%	0%	100%
		5	5	0%	100%	0%	100%
	D-ary Trees	2	7	0%	80%	0%	80%
		3	7	20%	100%	20%	100%
		4	7	0%	100%	0%	100%
		5	7	0%	60%	0%	60%
		2	5	100%	100%	100%	100%
		3	5	100%	100%	100%	100%
		4	5	100%	100%	100%	100%
		5	5	100%	100%	100%	100%
		2	7	60%	100%	60%	100%
		3	7	100%	100%	100%	100%
		4	7	20%	100%	20%	100%
		5	7	60%	100%	60%	100%
	Split Trees	2	9	80%	100%	80%	100%
		3	9	0%	40%	0%	40%
		4	9	0%	20%	0%	20%
		5	9	0%	0%	0%	0%
		2	12	0%	20%	0%	20%
		3	12	0%	20%	0%	20%
		4	12	0%	0%	0%	0%
		5	12	0%	0%	0%	0%
		2	15	0%	20%	0%	20%

Table 8: Results for tree methods.

ID	Desc.	Trained On		Test-Force R_t		Test-Gen R_t	
		D	M	SR	ABB	SR	ABB
	Multi. M	2	[2-5]	100%	100%	100%	100%
		3	[2-5]	100%	100%	100%	100%
		4	[2-5]	100%	100%	100%	100%
		5	[2-5]	100%	100%	100%	100%
		2	[2-7]	60%	60%	60%	60%
		3	[2-7]	20%	80%	20%	80%
		4	[2-7]	0%	60%	0%	60%
		5	[2-7]	0%	100%	0%	100%
		2	[2-9]	40%	40%	40%	40%
		3	[2-9]	20%	40%	20%	40%
		4	[2-9]	0%	20%	0%	20%
		5	[2-9]	0%	20%	0%	20%
		2	[2-12]	0%	20%	0%	20%
	Multi. D	[2-3]	5	0%	0%	0%	0%
		[2-4]	5	0%	0%	0%	0%
		[2-5]	5	0%	0%	0%	0%
	Multi. M with Multi. D	[2-3]	[2-9]	60%	60%	60%	60%
		[2-4]	[2-9]	20%	100%	20%	100%
		[2-5]	[2-9]	0%	100%	0%	100%
		[2-3]	[2-12]	0%	20%	0%	20%
	Multi. M with General Single Target	2	[2-9]	60%	100%	60%	100%
		3	[2-9]	0%	100%	0%	100%
		4	[2-9]	20%	80%	20%	80%
		5	[2-9]	80%	100%	80%	100%
		3	[2-12]	20%	80%	20%	80%

Table 9: All results are evaluated with the query being only the final node in the arm.