
Looking beyond the next token

Anonymous Authors¹

Abstract

The structure of causal language model training assumes that each token can be accurately predicted from the previous context. This contrasts with humans’ natural writing and reasoning process, where goals are typically known before the exact argument or phrasings. While this mismatch has been well studied in the literature, the working assumption has been that architectural changes are needed to address this mismatch. We argue that rearranging and processing the training data sequences can allow models to more accurately imitate the true data-generating process, and does not require any other changes to the architecture or training infrastructure. We demonstrate that this technique TRELAWNEY and the inference algorithms derived from it allow us to improve performance on benchmarks that span planning, algorithmic reasoning, and story generation tasks. Finally, our method naturally enables the generation of long-term goals at no additional cost. We investigate how using the model’s goal-generation capability can further improve long horizon planning and reasoning.

1. Introduction

Next-token prediction (NTP) is the primary objective for training sequence models. This objective involves a technique called *teacher forcing* (Williams & Zipser, 1989), where the model’s predicted output at each step is replaced with the ground truth from the real dataset. One of teacher forcing’s benefits is that it accelerates the training by providing the model with the correct previous output, so the learning does not suffer from error accumulation, and the gradient update is more stable. Another crucial benefit is that it enables parallelism and hardware acceleration in training because the model can simultaneously process all time steps, rather than sequentially waiting for its own predictions. However, Bachmann & Nagarajan (2024) argue that models trained with teacher forcing often fail to learn long-range dependencies, latching onto local patterns and surface-level correlations instead.

Several recent methods have been proposed to alleviate the issues of teacher forcing. One popular approach is *multi-*

token prediction, where the model learns to predict multiple tokens at the same time (Bachmann & Nagarajan, 2024; Gloeckle et al., 2024; Deepseek et al., 2024). Another family of approaches involves modifying the training objective to predict both the next token for a prefix and the previous token for a suffix by modifying the model architecture (Hu et al., 2025). Most of these approaches either involve non-trivial modification to the model architecture or make the learning process much harder by forcing the model to predict multiple tokens at the same time.

In this work, we investigate a data-centric approach to address these limitations. Instead of modifying the model architecture, our method TRELAWNEY modifies the training data to embed inductive biases directly. Concretely, we augment the training corpus by interleaving it with special lookahead tokens - $\langle T \rangle$ and $\langle /T \rangle$ that encapsulate future information (see Figure 1). The placement and content of these tokens can be determined either randomly or with task-specific knowledge. We hypothesize that this augmentation makes learning the long-term dependencies easier and imbues the model with the capacity to plan ahead. Furthermore, these modified training data naturally teach the model to guide the generation towards the future information, so the lookahead tokens can also let users exert fine-grained control over the long-term generation.

This work contributes a simple data-rearrangement procedure, TRELAWNEY, which results in both improved task performance in domains otherwise difficult for models trained with next token prediction, by decoupling the discussion of the training objective from the underlying data generating function the model needs to learn. Results are presented on planning, reasoning tasks and conditional story generation.

2. Preliminaries

Consider a sequence of tokens $\mathbf{y} = (y_1, y_2, \dots, y_T)$, where each token y_t belongs to a fixed vocabulary V . Suppose that $\mathbf{y}$ follows a distribution $P(\mathbf{y})$, an auto-regressive model P_θ factorizes the joint probability of $\mathbf{y} = (y_1, y_2, \dots, y_T)$ as follows: $P_\theta(\mathbf{y}) = \prod_{t=1}^T p_\theta(y_t \mid \mathbf{y}_{<t})$ where $\mathbf{y}_{<t} = (y_1, \dots, y_{t-1})$ denotes all tokens before index t .

Next token prediction. We train the model parameters θ to maximize the likelihood of each token under the ground truth context (teacher forcing) from the training corpus. If

our training data consists of N sequences $\{\mathbf{y}^{(i)}\}_{i=1}^N$, each of length T , then $\text{MLE}(\theta)$ is obtained by:

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^N \sum_{t=1}^T \log p_{\theta}(y_t^{(i)} | \mathbf{y}_{< t}^{(i)}).$$

Inference. At inference time, the model predicts the next token by either sampling or selecting the most likely token, conditioned on an optional context c (e.g., question).

2.1. Pitfalls of teacher forcing

Bachmann & Nagarajan (2024) highlighted the following subtle failure modes in teacher-forced training:

Clever Hans Cheat. When training with teacher-forcing, the model is provided with ground truth prefixes (e.g., $v_{\text{start}}, v_1, \dots, v_{i-1}$) that include parts of the answer. This extra information can enable the model to “cheat” by simply copying the easy tokens that follow without learning the true underlying plan.

Indecipherable Token Problem. Because the later tokens can be easily predicted using the Clever Hans cheat, the crucial early decision receives insufficient gradient signal. This early token becomes “indecipherable” since its correct prediction relies on long-range planning that is effectively bypassed during teacher-forced training.

Exposure bias. During inference, the model would likely make a mistake because the model has not learned the indecipherable token. The mismatch between training (where the model always sees the correct previous tokens) and inference (where it must rely on its own predictions) can lead to a cascading sequence of errors.

3. TRELAWNEY

Standard decoder-only transformers typically learn the factorization of a sequence defined in Equation 2. We introduce a data augmentation scheme that modifies the given sequence $\mathbf{y}$ as follows: first select a point d and insert a sequence of k tokens, $\mathbf{z} = (z_1, z_2, \dots, z_k)$, delimited with special tokens $\langle T \rangle$ and $\langle /T \rangle$. Concretely, we have the following augmentation: $(y_1 y_2 \dots y_T) \Rightarrow (y_1 y_2 \dots y_d \langle T \rangle \mathbf{z} \langle /T \rangle y_{d+1} \dots y_{T-1} y_T)$. The choices of d , k , and the content of $\mathbf{z}$ are flexible, and we present several strategies.

3.1. Augmentation schemas

Copying. We can directly copy a part of the sequence from a point after y_d to between the special tokens. For s such that $d < s \leq T - k$, $\mathbf{z}$ is the subsequence $\mathbf{y}_{s:s+k}$, resulting in $\tilde{\mathbf{y}}_{\text{copy}} \equiv y_1 y_2 \dots y_d \langle T \rangle \mathbf{y}_{s:s+k} \langle /T \rangle y_{d+1} \dots y_{T-1} y_T$. The choice of $\mathbf{z}$ can have a significant impact on the behavior of the resulting model. For example, for certain types of data, there are *decision points* where there are many different possible futures. These points are good candidates for choosing d . Analogously, we can choose $\mathbf{z}$ to be *future tokens*

that indicate which future is being generated. These terms are intentionally defined loosely since the design space is large (§4.1, §A.1). Without this prior information, we can also select d , s , and k randomly, which can be useful for generic language modeling.

Positional information. In the previous approach, d and s can vary between different data points. This can be problematic if two sequences have very different values of $s - d$. Intuitively, this makes the modeling task harder because there may be conflicting information between different sequences. For example, suppose $\mathbf{y}^1$ and $\mathbf{y}^2$ share the same prefixes, $\mathbf{y}_{:d}^1 = \mathbf{y}_{:d}^2$ but the relevant future tokens are at locations with large differences. To mitigate this conflict, we introduce additional *positional information* into the future tokens, $\zeta(k, \mathbf{z})$. For example, we can have: $\zeta(k, \mathbf{z}) = \text{“I want the [k]th sentence from here to be } \mathbf{z}''$. $\tilde{\mathbf{y}}_{\text{copy+pos}} \equiv y_1 y_2 \dots y_d \langle T \rangle \zeta(k, \mathbf{z}) \langle /T \rangle y_{d+1} \dots y_{d+k} \dots y_n$

The exact design of the positional information can be problem-dependent (§4.2). It is also possible to introduce various agentic behaviors via ζ , though we only conduct a preliminary investigation in this direction.

3.2. Dataset Construction and Training Objective

Dataset construction. Our goal is to introduce additional capabilities via the augmentation schema shown above. However, it is desirable to do so without hurting the traditional language modeling ability of the model. To do so, we train on both regular text and augmented text simultaneously. Given an original dataset $D = \{\mathbf{y}^{(i)}\}_{i=1}^N$ and an augmentation schema aug , we can construct a distribution for the original dataset, and a distribution for the augmented dataset: $\mathcal{D}(\mathbf{s}) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}\{\mathbf{s} = \mathbf{y}^{(i)}\}$, $\mathcal{D}_{\text{aug}}(\mathbf{s}) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}\{\mathbf{s} = \text{aug}(\mathbf{y}^{(i)})\}$. For a probability p that controls how much of the training distribution comprises the original data, the training distribution is the following mixture: $\mathcal{D}'(\mathbf{s}) = p \mathcal{D}(\mathbf{s}) + (1 - p) \mathcal{D}_{\text{aug}}(\mathbf{s})$.

Training and loss function. During training, the model parameters are optimized using a standard cross-entropy loss with teacher forcing on $\mathcal{D}'$. This allows us to take advantage of all existing engineering optimizations for training language models. One caveat for training with the new dataset $\mathcal{D}'$ is that choosing the decision point and future tokens arbitrarily will result in a large portion of sequences with the next token being $\langle /T \rangle$ at arbitrary locations. This would distract from the learning process and does not help learning the underlying distribution, since the special tokens are synthetically introduced. Instead, we modify the regular cross-entropy loss by masking the special start token, $\langle T \rangle$: $\mathcal{L}(\mathcal{D}') = -\mathbb{E}_{\mathbf{y} \sim \mathcal{D}'} \left[\frac{1}{|\mathbf{y}|} \sum_{j=1}^{|\mathbf{y}|} \mathbb{I}\{y_j \neq \langle T \rangle\} \log P(y_j | \mathbf{y}_{< j}) \right]$. Here, $\mathbb{I}\{y_j \neq \langle T \rangle\}$ ensures no loss is computed for the prediction of the special token $\langle T \rangle$. We do not exclude the loss on

$\langle /T \rangle$ because there is a utility to predicting the closing of the future tokens, which we will elaborate on below.

3.3. Inference

Standard autoregressive generation. The model generates sequences autoregressively without any intervention, following any standard decoding algorithm.

$\langle T \rangle$ -generation. We aim to enable the model to explicitly consider future context at appropriate decision points, to improve its ability to plan ahead. At each decision point y_d in sequence generation, we explicitly insert the special token $\langle T \rangle$. Subsequently, **(a)** either the model generates the sequence $\mathbf{z}$ autonomously, enabling it to create plausible future plans, or **(b)** incorporates a user-specified sequence $\mathbf{z}$, enhancing controllability. Recall that during the training process, we compute the loss on the $\langle /T \rangle$ token, this allows the model to generate future goals, which can then be used for conditional generation.

4. Experiments

We hypothesize that prioritizing predictions of challenging tokens or strategic goals can improve generation quality. To test this, we conduct experiments across three synthetic and natural language tasks. Synthetic tasks serve as a controlled environment where specific token-level or semantic unit challenges - otherwise difficult to disentangle in natural language - can be precisely studied.

We begin with two synthetic benchmarks: the star graph (§ 4.1), a didactic example used to highlight limitations of next-token prediction even in simple tasks, and an algorithmic reasoning benchmark (§ A.1), which allow us to test whether explicitly adding future goals or anchor points improves performance in simple reasoning tasks. Finally, we extend our analysis to story generation (§ 4.2) to assess controllable generation and planning. Our experiments are designed to answer the following questions: (1) Does TRELAWNEY enable the generation of long-term goals? (2) Does TRELAWNEY training improve planning when using autoregressive inference? (3) Do the goals generated explicitly by the model improve planning? (4) Does explicitly providing goal sequences $\mathbf{z}$'s improve controllability?

4.1. Star Graph

The star graph is a simple path-finding problem introduced by Bachmann & Nagarajan (2024), where, given a directed graph $G(d, n)$ with degree d and path length n , the objective is to find a path from the start node to the goal node (Fig. 2). Despite its simplicity, traditional NTP struggles on this task. A key challenge is that the critical decision point occurs at v_1 , the first node after v_{start} . This node is hard to predict because v_{start} has many outgoing edges.

Dataset and Augmentation Schema. To mitigate these issues, we introduce a future subgoal $\mathbf{z}$, as any contiguous

subsection of the path in $[v_2, v_{\text{goal}}]$. This modification compels the model to generate a meaningful intermediate plan rather than simply copying the full ground truth prefix. As a result, the model receives a stronger learning signal for critical early decision-making. Each example $\mathbf{y} = (\mathbf{p}, \mathbf{c})$ in the dataset is a prefix and completion pair. The prefix $\mathbf{p}$ is given by the adjacency list of G followed by the $v_{\text{start}}, v_{\text{goal}} =$. The completion $\mathbf{c}$ is the path $v_{\text{start}}, v_1, v_2, \dots, v_{\text{goal}}$, i.e., $\mathbf{p} \equiv \text{Adj}(G) \mid v_{\text{start}}, v_{\text{goal}} =$ and $\mathbf{c} \equiv v_{\text{start}}, v_1, v_2, \dots, v_{\text{goal}}$.

Our augmentation schema $\mathbf{y} \Rightarrow \tilde{\mathbf{y}}_{\text{copy}}$ for this task is as follows:

$$\mathbf{y} \equiv \text{Adj}(G) \mid v_{\text{start}}, v_{\text{goal}} = v_{\text{start}}, v_1, v_2, \dots, v_{\text{goal}}$$

$$\tilde{\mathbf{y}}_{\text{copy}} \equiv \text{Adj}(G) \mid v_{\text{start}}, v_{\text{goal}} = v_{\text{start}}, \langle T \rangle \mathbf{z}, \langle /T \rangle v_1, \dots, v_{\text{goal}}$$

Choice of $\mathbf{z}$. We vary $\mathbf{z}$ (a contiguous subsequence of future tokens) across experiments and ablations (see Figure 2). Its role is to guide planning by indicating a subgoal on the path from v_{start} to v_{goal} . We exclude v_1 to avoid the Clever Hans cheat discussed above. We exclude v_{goal} so that the model learns the long-term dependency between start and goal without having direct access to the goal token.

Training. Data for all experiments are generated programmatically using the official implementation. Although we use pretrained models, each node remains a single token in the tokenizer. All models are trained on 200,000 examples as described in Appendix A.4. We follow standard teacher forcing training with two variants of augmentation schemas.

TRELAWNEY-fixed: In a single training run, the choice of $\mathbf{z}$ is fixed across all examples. Specifically, $\mathbf{z}$ is chosen as a contiguous sequence of 1 to 4 nodes with a fixed start and end point across all sequences in the dataset (Figure 2).

TRELAWNEY-random: $\mathbf{z}$ can vary between examples. We randomly select any contiguous subsequence of the path after v_1 to serve as $\mathbf{z}$ in $\tilde{\mathbf{y}}_{\text{copy}}$. We do not include v_1 (the hard node) as part of $\mathbf{z}$ (Figure 2). Without fixed positional information, the model learns to generate its own goals of varying lengths. We observe that this variant is successful in solving longer planning problems.

Evaluation. We evaluate the models on 5,000 held-out examples for each graph, reporting the accuracy of the generated path compared to the ground truth. For the next-token prediction baseline, we evaluate the model using standard autoregressive generation. For models trained with TRELAWNEY, we assess both standard autoregressive and $\langle T \rangle$ -generation. In the conditional setting, the model uses either model-generated $\mathbf{z}$'s as goals or user-provided ground truth "future goals" as hints. Standard autoregressive generation allows us to test whether TRELAWNEY improves regular generation. $\langle T \rangle$ -generation demonstrates whether the model has learned to generate plausible future goals and

		Path planning $G(*,*)$			
		$G(2,5)$	$G(5,5)$	$G(20,5)$	$G(2,10)$
AutoReg.	NTP	<u>0.50</u>	<u>0.20</u>	<u>0.05</u>	0.50
	TRELAWNEY				
	– Fixed	1.00	1.00	1.00	0.52
	– Random	1.00	1.00	1.00	0.50
Gen.	TRELAWNEY				
	– Fixed	1.00	1.00	1.00	0.57
	– Random	1.00	1.00	1.00	<u>0.91</u>
Spec.	TRELAWNEY				
	– Fixed	1.00	1.00	1.00	1.00
	– Random	1.00	1.00	1.00	<u>0.91</u>

Table 1: TRELAWNEY outperforms next token prediction in Path planning (*Star Graph*) Autoreg, Gen. and Spec. refer to inference methods of standard autoregressive generation, model generated $\langle T \rangle$ -generation and user specified $\langle T \rangle$ -generation respectively. Fixed, Random and Rule-Based indicate the data augmentation strategies used in training. **Bold** is the best and underline is the second best.

use these goals for better planning. By providing intermediate hints, we evaluate if the model can leverage these cues to solve the larger planning problems.

Results. On shorter graphs ($n=5$), training with TRELAWNEY improves autoregressive generation at no additional cost, suggesting that the model implicitly learns to plan better and can generate long-term goals. For longer graphs $G(2, 10)$, TRELAWNEY-random can complete the task when the model is used to generate its own subgoal sequence z , indicating that model-generated goals can improve planning. TRELAWNEY-random is notably more performant on graphs with longer paths when compared to TRELAWNEY-fixed. We hypothesize that the random variant’s flexibility in future goal selection has a regularizing effect that allows the model to learn better representations. Both variants of TRELAWNEY succeed when user-provided goal sequences are provided, showing that explicit goal hints allow for better controllability. (See: Tab. 1) Ablations conducted on larger models (A.6) show that the ability to plan for future tokens improves with increase in model capacity.

4.2. Natural Language Planning

We evaluate our approach on the **Tiny Stories** dataset (Eldan & Li, 2023). In this task, the aim is to generate coherent stories conditioned on specified goals. This benchmark tests whether our strategy of inserting future tokens can enhance planning in natural language generation.

Dataset and Augmentation Schema. Each example $y = (p, c)$ is a prefix-completion pair. We use a sentence parser to segment each story into individual sentences or phrases. If a story is split into sentences $s_1, s_2, \dots, s_n$, the prefix p is the beginning of the story (e.g., $s_1 s_2$) and

the completion c is the remainder (i.e., $s_3 s_4 \dots s_n$). Our augmentation schema $y \Rightarrow \tilde{y}_{\text{copy+pos}}$ is defined as:

$$y \equiv s_1 s_2 \dots s_n$$

$$\tilde{y}_{\text{copy+pos}} \equiv s_1 s_2 \dots s_d \langle T \rangle \zeta(k, s_{d+k}) \langle /T \rangle s_{d+1} \dots s_{d+k} \dots s_n$$

$$\zeta(k, s) = \text{“I want the } [k]\text{-th sentence from here to be”}$$

Choice of $\zeta(k, s)$. We choose decision points randomly at the end of the k -th sentence in the document, as the position to insert $\zeta(k, s)$. The subgoal $[s]$ is defined in $\zeta(d, s)$ as extracted from the corresponding sentence s_{d+k} .

Training. All models are trained on 300,000 examples from the Tiny Stories dataset. (See App. A.4)

- *Next-token prediction:* We follow standard teacher forced training on the dataset $\mathcal{D}$.
- *TRELAWNEY-implicit-pos:* We insert goals into $\mathcal{D}_{\text{aug}}$ by specifying how far ahead the goal is in natural language. This probes if the model can discern positional information without explicit labels.

Evaluation: Following the evaluation protocol in Hu et al. (2025), we use GPT-4 as a judge to rate 100 generated stories from each model. The stories are anonymized and shuffled to prevent any information leakage about the author before being passed to the judge. Each evaluation is repeated over 6 trials. We report the win rate with binomial confidence intervals computed at a 95% significance level.

RQ1: Does TRELAWNEY improve goal reaching ability i.e., resulting in more controllable generation? We compare the completions from few-shot prompts on the baseline with those obtained by explicitly specifying goals on TRELAWNEY. Qualitatively, models trained with TRELAWNEY generate stories that more effectively reach the intended long-term goals (see Figure 1). Quantitatively, GPT-4 prefers TRELAWNEY to few-shot prompts on next-token-prediction, 76.53% of the time, with a confidence interval of [72.9%, 79.9%]; which suggests that TRELAWNEY is more effective at controllable generation than few-shot prompting. (See : D.2)

RQ2: Preference on stories generated by standard autoregressive generation. We compare the standard autoregressive generations from models trained with TRELAWNEY and models trained with NTP. Quantitatively, we observe that GPT-4 prefers autoregressive generations on TRELAWNEY to next-token-prediction, 40.35% of the time, with a binomial confidence interval of [44.5%, 36.2%]. For examples, see D.2

Ablations. We evaluate perplexity on Wikitext to verify that TRELAWNEY maintains language model performance, with results comparable to the baseline. (See: D.3)

5. Discussion

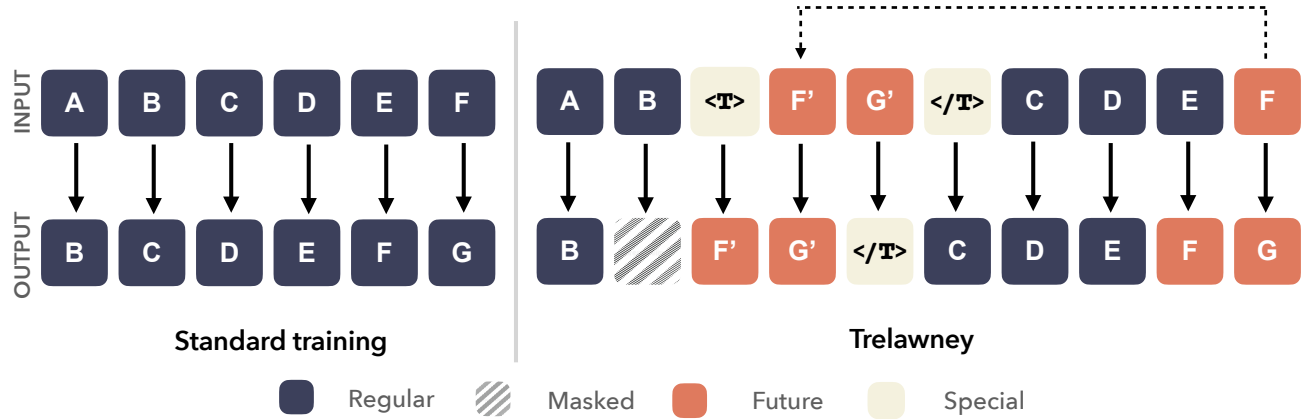
Our small-scale experiments are effective but limited to synthetic or constrained scenarios. Our initial results suggest that adaptation to more general settings may be useful.

References

- Arora, K., Asri, L. E., Bahuleyan, H., and Cheung, J. C. K. Why exposure bias matters: An imitation learning perspective of error accumulation in language generation. *arXiv preprint arXiv:2204.01171*, 2022.
- Bachmann, G. and Nagarajan, V. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*, 2024.
- Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., and Chen, M. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.
- Bigelow, E., Holtzman, A., Tanaka, H., and Ullman, T. Forking paths in neural text generation. *arXiv preprint arXiv:2412.07961*, 2024.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.
- Dathathri, S., Madotto, A., Lan, J., Hung, J., Frank, E., Molino, P., Yosinski, J., and Liu, R. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.
- Deepseek, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- Du, L., Mei, H., and Eisner, J. Autoregressive modeling with lookahead attention. *arXiv preprint arXiv:2305.12272*, 2023.
- Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.
- Frydenlund, A. The mystery of the pathological path-star task for language models. *arXiv preprint arXiv:2410.13779*, 2024.
- Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.
- Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022.
- Goyal, S., Ji, Z., Rawat, A. S., Menon, A. K., Kumar, S., and Nagarajan, V. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.
- Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.
- Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- Han, T., Xie, W., and Zisserman, A. Video representation learning by dense predictive coding. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pp. 0–0, 2019.
- Hu, E. S., Ahn, K., Liu, Q., Xu, H., Tomar, M., Langford, A., Jayaraman, D., Lamb, A., and Langford, J. The belief state transformer. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=ThRMTCgpvo>.
- Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.
- Kitouni, O., Nolte, N. S., Williams, A., Rabbat, M., Bouchacourt, D., and Ibrahim, M. The factorization curse: Which tokens you predict underlie the reversal curse and more. *Advances in Neural Information Processing Systems*, 37: 112329–112355, 2024.
- Krause, B., Gotmare, A. D., McCann, B., Keskar, N. S., Joty, S., Socher, R., and Rajani, N. F. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.
- LeCun, Y. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.
- Lin, J., Du, Y., Watkins, O., Hafner, D., Abbeel, P., Klein, D., and Dragan, A. Learning to model the world with language.
- Lin, Z., Gou, Z., Gong, Y., Liu, X., Shen, Y., Xu, R., Lin, C., Yang, Y., Jiao, J., Duan, N., et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.
- Markeeva, L., McLeish, S., Ibarz, B., Bounsi, W., Kozlova, O., Vitvitskyi, A., Blundell, C., Goldstein, T., Schwarzschild, A., and Veličković, P. The clrs-text algorithmic reasoning language benchmark. *arXiv preprint arXiv:2406.04229*, 2024.
- Nolte, N., Kitouni, O., Williams, A., Rabbat, M., and Ibrahim, M. Transformers can navigate mazes with multi-step prediction. *arXiv preprint arXiv:2412.05117*, 2024.

- Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. 2021.
- Pannatier, A., Courdier, E., and Fleuret, F. σ -gpts: A new approach to autoregressive models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 143–159. Springer, 2024.
- Qi, W., Yan, Y., Gong, Y., Liu, D., Duan, N., Chen, J., Zhang, R., and Zhou, M. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *arXiv preprint arXiv:2001.04063*, 2020.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.
- Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.
- Tarjan, R. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.
- Vondrick, C., Pirsaviash, H., and Torralba, A. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 98–106, 2016.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- Welleck, S., Brantley, K., Iii, H. D., and Cho, K. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pp. 6716–6726. PMLR, 2019.
- Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.
- Wu, W., Morris, J. X., and Levine, L. Do language models plan ahead for future tokens? In *First Conference on Language Modeling*.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.
- Yin, Y., Ding, J., Song, K., and Zhang, Y. Semformer: Transformer language models with semantic planning. *arXiv preprint arXiv:2409.11143*, 2024.

A. Appendix



PROMPT Once upon a time, there was a little girl named Lily. Lily loved to play in the rain.

FUTURE

<T> I want the [2]-nd sentence from here to be: "Hello little frog! </T>

<T> I want the [4]-th sentence from here to be: "Hello little frog! </T>

<T> I want the [8]-th sentence from here to be: "Hello little frog! </T>

GENERATION

One day, Lily was playing in the rain and she saw a little frog. "Hello little frog! What are you doing here?" Lily asked. The frog replied, "I am looking for a friend to play with. Can you be my friend?" Lily was happy to have a new friend. She said, "Yes, I can be your friend. We can play in the rain together." So, the frog and Lily played in the rain, making puddles and having fun. At the end of the day, they said goodbye and promised to play again the next day.

One day, Lily went outside to play. She saw a big frog on a log. The frog looked sad. "Hello little frog! Why are you sad?" Lily asked. The frog replied, "I can't jump high. I need help." Lily thought for a moment and said, "I have an idea! Let's play together and I will help you jump high." The frog was so happy and thanked Lily. They jumped high and played all day. From that day on, Lily and the frog became best friends.

One day, Lily went outside to play in the rain. She jumped in the puddles and laughed. She loved the feeling of the rain on her face. Suddenly, Lily saw a frog in a puddle. The frog was small and wet. Lily wanted to help the frog. She picked up the frog and put it on a rock. "Hello little frog! You are so wet and cold," said Lily. The frog smiled and thanked Lily for helping. From that day on, Lily and the frog became good friends.

Figure 1: TRELAWNEY. **(Top)** We modify the training data by inserting tokens from the future, delimited with special tokens <T> and </T>, into the modified sequences. By first predicting tokens in the future (i.e., F' G'), and the path leading to the future (i.e., CDE) as well as the actual future (i.e., FG) will be easier to predict. **(Bottom)** Illustration of TRELAWNEY's effect during generation. The top is the prompt, and the middle is different future tokens. The generations are coherent and read naturally.

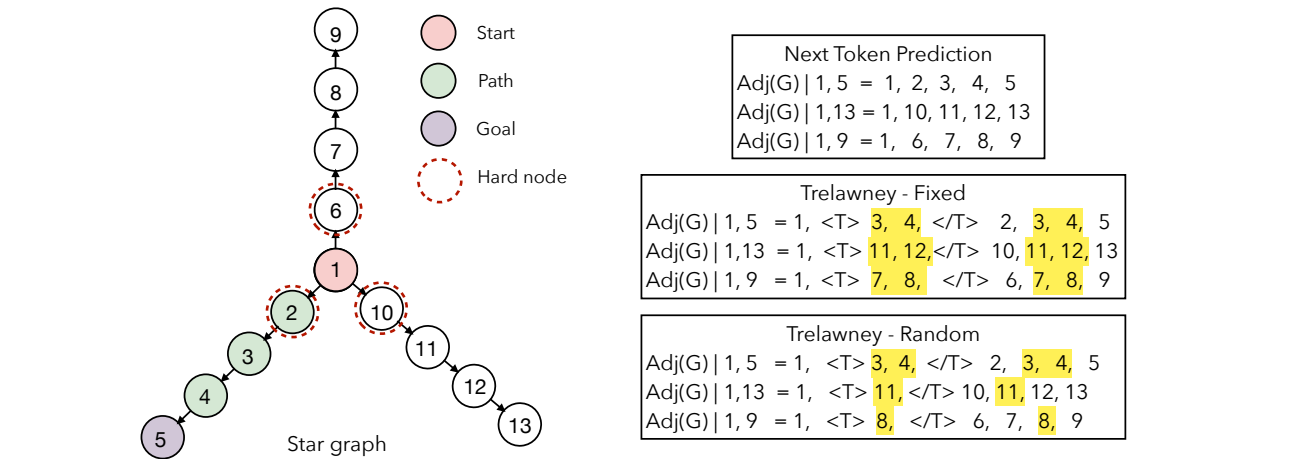


Figure 2: In the star graph, there are key “hard nodes” that indicate the moment of branching, after which the path and goal become clear. Above is a visualization of the construction of $\mathcal{D}'$ for the star graph with corresponding linearizations.

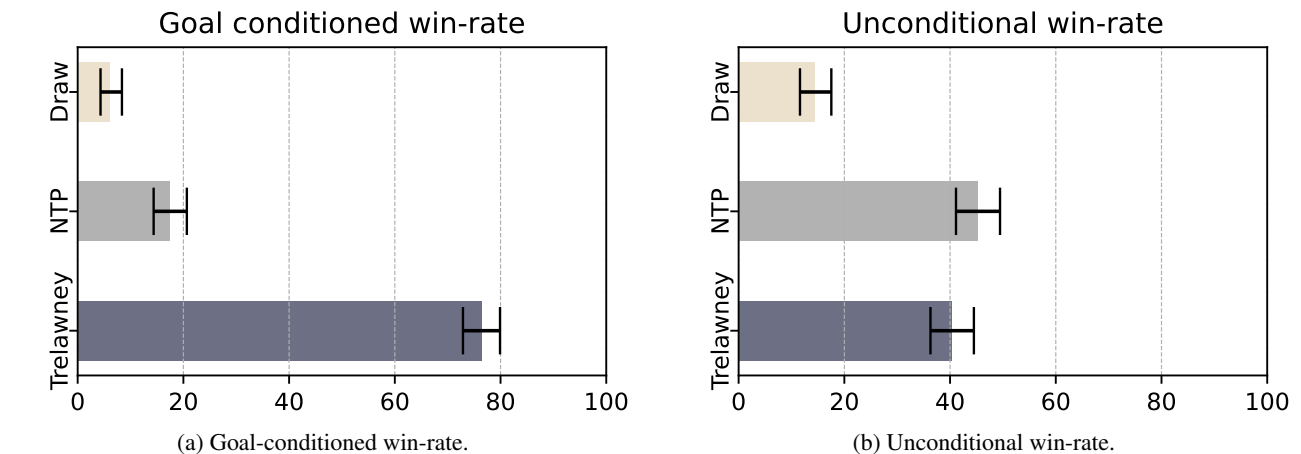


Figure 3: Our evaluation of story generation demonstrates greatly improved performance when goal-conditioned, without hurting the performance of unconditional generation.

A.1. Additional synthetic tasks - Algorithmic Reasoning

CLRS-Text (Markeeva et al., 2024)¹ is a benchmark of algorithmic reasoning. The input is the algorithm name, followed by a step-by-step reasoning trace and the final answer. We pick a representative example from algorithms that require backtracking, i.e., tasks that benefit from information of future states. We choose strongly-connected-components, a step-by-step sequential prediction task where each step is longer than one token, and report results on it. The trace contains the execution of Tarjan’s algorithm (Tarjan, 1972), which computes strongly connected components in linear time by performing a depth-first search that tracks low-link values and uses a stack to detect cycles.

Dataset and Augmentation Schema. In each example $\mathbf{y} = (\mathbf{p}, \mathbf{c})$ of the strongly-connected-components subset, the prefix $\mathbf{p}$ is given by the adjacency matrix of the initial graph. The completion $\mathbf{c}$ is graph execution traces of the algorithm followed by the final answer, i.e., $\mathbf{p} \equiv \text{Adj}(G)$ and $\mathbf{c} \equiv t_1, t_2 \dots t_n | F$ where t_i is the state of the graphical trace and F is the final answer. Our augmentation schema $\mathbf{y} \implies \tilde{\mathbf{y}}_{\text{copy}}$ for this task is as follows:

$$\begin{aligned} \mathbf{y} &\equiv \text{algo: Adj}(G) = t_1, t_2, \dots, t_n | F \\ \tilde{\mathbf{y}}_{\text{copy}} &\equiv \text{algo: Adj}(G) = t_1, \langle T \rangle \mathbf{z} \langle /T \rangle t_2, \dots, t_n | F \end{aligned}$$

Unlike the star graph task — where failure typically occurs at a single critical decision point — the algorithmic reasoning tasks involve multiple branching points where errors can accumulate. In the strongly connected components subset, the state sequence t represents the graph execution trace and comprises multiple tokens, each corresponding to a distinct graph state. By segmenting the trace into these meaningful units, our augmentation schema is better able to capture intermediate reasoning steps and guide the model’s planning process throughout the entire execution trace.

Choice of $\mathbf{z}$. For simplicity, we fix the decision point y_d at the second state in each trace. Unlike the star graph task, where the hard tokens are easy to extract, algorithmic reasoning tasks do not present a clear failure point — there can be many points in the trace at which misprediction causes the entire generation to diverge. We only pick $\mathbf{z}$ as a complete step t_i in the trace and how i is determined for each variant.

Training. Data for all experiments are sub-selected from the original dataset. We train a single model on problems of varying sizes. Since we do not test for length generalization, we only report accuracies on problem sizes present in the training corpus. For strongly connected components, all models are trained on 60,000 examples. We train two variants (See C.1) of our method:

TRELAWNEY-rule-based: For every example in $\mathcal{D}_{\text{aug}}$, $\mathbf{z}$ is chosen as the first change in the trace provided. The position of $\mathbf{z}$ in the trace varies across graph sizes and graphs.

TRELAWNEY-random: $\mathbf{z}$ is chosen as a single random state in the trace provided.

		Alg Reasoning scc-				
		scc-4	scc-5	scc-11	scc-12	scc-15
AutoReg.	NTP	1.00	0.99	0.62	0.57	0.27
	TRELAWNEY					
	– Fixed	1.00	1.00	0.73	0.62	0.31
	– Random	1.00	0.978	0.718	0.706	0.476
Generated	NTP	–	–	–	–	–
	TRELAWNEY					
	– Fixed	1.00	1.00	0.73	0.65	0.34
	– Random	1.00	<u>0.998</u>	0.776	<u>0.79</u>	<u>0.512</u>
Specified	NTP	–	–	–	–	–
	TRELAWNEY					
	– Fixed	1.00	1.00	0.84	0.76	0.47
	– Random	1.00	<u>0.998</u>	<u>0.828</u>	0.812	0.544

Table 2: Algorithmic-reasoning performance on SCCs.

Evaluation. We evaluate on 500 examples (CLRS-Text-test)² per graph length. We evaluate the models similar to the

¹<https://huggingface.com/datasets/tomg-group-umd/CLRS-Text-train>

²<https://huggingface.co/datasets/tomg-group-umd/CLRS-Text-test>

star-graph setting, and report the accuracies of the final answer.

Results Results from Figure 5 show a trend that TRELAWNEY-Random consistently improves on next token prediction when using $\langle T \rangle$ -generation and, surprisingly, in standard autoregressive generation as well. TRELAWNEY-Rule-Based although being chosen more strategically, performs worse than $\langle T \rangle$ -Random.

A.2. Related work

Next token prediction. Bachmann & Nagarajan (2024) characterizes two failure that occur in next-token prediction, those that emerge from (1) teacher-forced training, and (2) those emerging at inference, where errors compound (the so-called snowballing failure). Much of the prior work (Arora et al., 2022; Ross et al., 2011) has focused primarily on the inference-time errors. In contrast, during training, the maximum likelihood estimation (MLE) objective treats all tokens equally. However, Bigelow et al. (2024) provides empirical evidence that tokens contribute unequally to the overall performance, suggesting that some tokens are inherently more critical than others. Relatedly, Lin et al. (2024) propose leveraging a stronger model to identify and prioritize these important tokens for more efficient pretraining. Nye et al. (2021) introduce scratchpads which augment the model’s input with intermediate reasoning steps to support multi-step problem solving. Goyal et al. (2023) introduces pause tokens at training and inference, as a mechanism for delayed next-token prediction, which improves performance across several language tasks.

Architectural changes. (Du et al., 2023; Yin et al., 2024) propose architectural modifications to improve planning. Qi et al. (2020) predict multiple n-grams rather than a single token at a time. Similarly, Gloeckle et al. (2024); Deepseek et al. (2024) incorporate these insights and have empirically validated the approach at scale. Hu et al. (2025) introduce an additional encoder that learns representations for both the next and previous states simultaneously. Frydenlund (2024) explores alternate encoder-based architectures for planning.

Controllable generation. Our work is also related to the literature of controllable generation, where the models are conditioned to follow high-level goals or guidelines provided through explicit instructions or auxiliary inputs. Prominent methods include Keskar et al. (2019); Dathathri et al. (2019); Krause et al. (2020). More recent models can be controlled via *prompting* (Brown et al., 2020; Wei et al., 2022). In comparison, TRELAWNEY does not require a curated dataset or additional classifiers and achieves fine-grained temporal control.

Non-causal sequence modeling. offers an alternative to the traditional autoregressive, left-to-right generation constraint by allowing the model to use both past and future context (Gu et al., 2017; Gong et al., 2022; Nolte et al., 2024). Bavarian et al. (2022) propose a “fill in the middle” strategy which changes the data ordering, while T5 (Raffel et al., 2020) incorporates span corruption, σ -GPT (Pannatier et al., 2024) uses on-the-fly order modulation, MLM- $\mathcal{U}$ (Kitouni et al., 2024) uses uniform masking similar to the diffusion objective and XLNet (Yang et al., 2019) leverages permutation-based training. Inference-time strategies, such as tree generation (Welleck et al., 2019), have also been explored. Beyond language modeling, video prediction (Han et al., 2019; Vondrick et al., 2016) similarly relies on non-causal prediction of future frames or states. In control tasks and world modeling (LeCun, 2022; Hafner et al., 2023; Lin et al.), non-causal approaches provide a more comprehensive representation of environmental dynamics, thereby enhancing long-term planning.

A.3. Reproducibility statement

Code and datasets for all experiments are currently in preparation and will be released.

A.4. Implementation details

Training details: All results are reported on the pretrained-Llama 3.2-1B model. We conducted experiments by sweeping over learning rates of $1e-5$, $2e-5$, and $1e-6$, using the AdamW optimizer with a linear learning rate scheduler for one epoch, and reporting the best result. We use the masked cross-entropy loss specified in § 3.2. We use $p = 0.5$ for all experiments. All experiments were run on 4xA6000 GPUs or 4xL40S GPUs. We will also provide the full list of hyperparameters and release code and datasets used.

A.5. Ablations - Autoregressive architectures

In this section we also compare against other autoregressive architectures. We use mamba as a representative model class for state space models. We observe that using TRELAWNEY-Random improves on next token prediction on state space architectures as well.

		Path planning $G(*,*)$			
		$G(2,5)$	$G(5,5)$	$G(20,5)$	$G(2,10)$
AR:	NTP	0.50	0.20	0.05	0.50
	TRELAWNEY	1.0	0.998	0.049	0.50
Gen:	NTP	–	–	–	–
	TRELAWNEY	1.0	0.997	0.048	0.511
Spec:	NTP	–	–	–	–
	TRELAWNEY	1.0	0.998	0.048	0.50

Table 3: Mamba-1.5B - Results on star graph

A.6. Ablations - Model sizing

To compare the effects of model size on TRELAWNEY-Random, we perform on 0.5B (Qwen2.5-0.5B), 1B (Llama-3.2-1B) and 3B (Llama-3.2-3B) models. We do not account for architectural differences between the Qwen 0.5B model and the 1B and 3B Llama models.

The smallest model is unable to solve the longest graph that we test for $G(2, 10)$, while the 1B model is able to solve the graph when allowed to generate z . Finally, the 3B model, is able to solve the graph with only autoregressive generation when trained with TRELAWNEY. This hints at TRELAWNEY being more effective on larger models, potentially learning better representations, and being easily scalable. Interestingly, larger models can solve the simplest graphs ($G(2, 5)$, $G(5, 5)$) autoregressively. We speculate that this could be due to pre-caching improving with scale as previously observed by (Wu et al.)

		Path planning $G(*,*)$			
		$G(2,5)$	$G(5,5)$	$G(20,5)$	$G(2,10)$
AR:	NTP	0.50	0.20	0.05	0.50
	TRELAWNEY	1.0	1.0	0.874	0.533
Gen:	NTP	–	–	–	–
	TRELAWNEY	1.0	1.0	0.847	0.514
Spec:	NTP	–	–	–	–
	TRELAWNEY	1.0	1.0	0.931	0.523

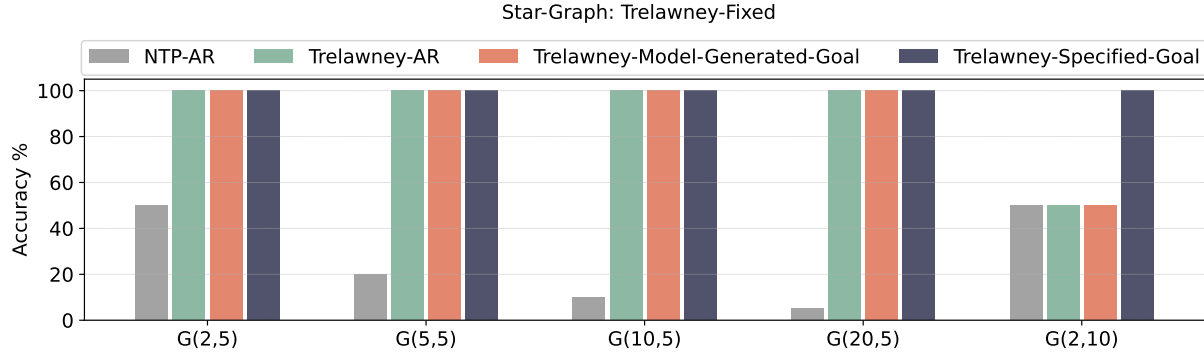
Table 4: Qwen/Qwen2.5-0.5B

		Path planning $G(*,*)$			
		$G(2,5)$	$G(5,5)$	$G(20,5)$	$G(2,10)$
AR:	NTP	1.0	1.0	0.05	0.50
	TRELAWNEY	1.0	1.0	1.0	1.0
Gen:	NTP	–	–	–	–
	TRELAWNEY	1.0	1.0	1.0	1.0
Spec:	NTP	–	–	–	–
	TRELAWNEY	1.0	1.0	1.0	1.0

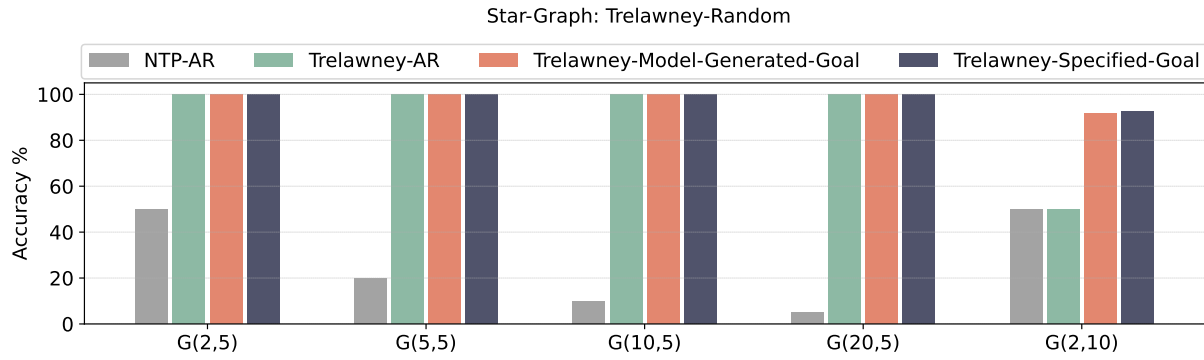
Table 5: meta-llama/Llama-3.2-3B

B. Star graph

B.1. Results - Llama-3.2-1B



(a) Accuracies TRELAWNEY-Fixed



(b) Accuracies TRELAWNEY-Random

Figure 4: Results on Star Graph.

C. Algorithmic reasoning

C.1. Examples

We provide examples of the data augmentation schema used in D_{aug} for the strongly connected components task. Looking at the examples in the data, we see many repeated states t in the trace. The rule used to pick z was likely more beneficial, since the first change in the trace state contains more information than a random trace state.

Strongly connected components - Examples

Prefix:

```
"strongly_connected_components:
  A: [[0 0 0 0 0 0],
      [0 0 0 0 0 0],
      [0 0 0 0 1 0],
      [0 0 0 1 1 0],
      [0 0 1 1 1 0],
      [0 0 0 0 0 1]],
  initial_trace: [0 1 2 3 4 5] trace | scc_id:",
```

Completion:

```
"[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 2 5], [0 1 2 2 2 5],
[0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5],
[0 1 2 2 2 5], [0 1 2 2 2 5] | [0 1 2 2 2 5]"
```

Completion for TRELAWNEY-Rule-Based:

```
"[0 1 2 3 4 5], <T> [0 1 2 3 2 5], </T> [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5],
[0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5] | [0 1 2 2 2 5]"
```

Completion for TRELAWNEY-Random:

```
"[0 1 2 3 4 5], <T> [0 1 2 3 4 5] </T> [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5],
[0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 4 5], [0 1 2 3 2 5],
[0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5],
[0 1 2 2 2 5], [0 1 2 2 2 5], [0 1 2 2 2 5] | [0 1 2 2 2 5]"
```


C.2. Results - Llama-3.2-1B

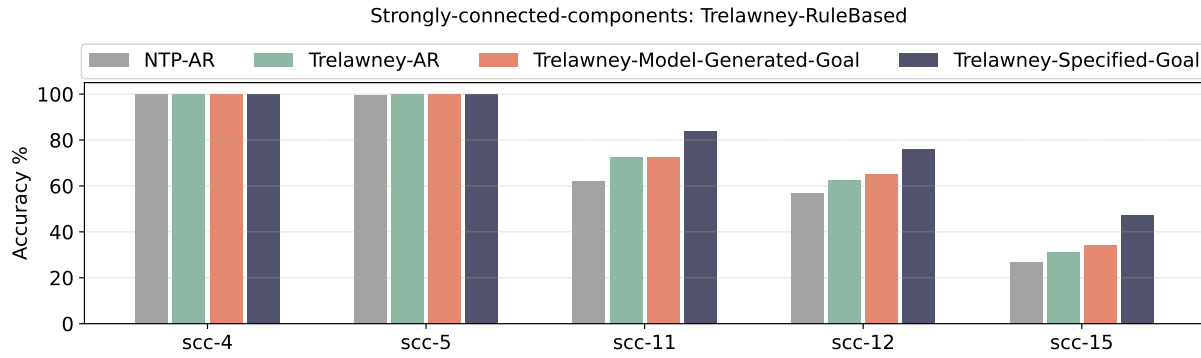


Figure 5: Accuracies - Strongly connected components TRELAWNEY-Rule-Based

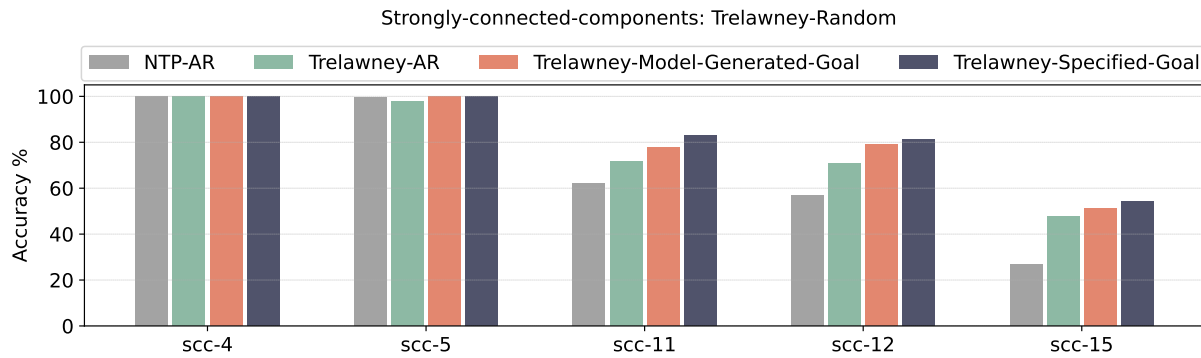


Figure 6: Accuracies - Strongly connected components TRELAWNEY-Random

D. Story generation

D.1. Examples

Once upon a time, there was a little girl named Lily. Lily loved to play in the rain. She would jump in all the puddles and get very wet. One day, Lily saw a cute little frog hopping around in the rain. "Hello little frog! What are you doing in the rain?" asked Lily. "I like the rain because it helps create more puddles for me to swim in," replied the frog. Lily smiled and said, "That's so cool! I love playing in the rain too." ...

Baseline

Once upon a time, there was a little girl named Lily. Lily loved to play in the rain. $\langle T \rangle$ I think [3] sentences away should be: "Hello little frog! $\langle /T \rangle$ [1] She would jump in all the puddles and get very wet. [2] One day, Lily saw a cute little frog hopping around in the rain. [3] "Hello little frog! [4] What are you doing in the rain?" asked Lily. [5] "I like the rain because it helps create more puddles for me to swim in," [6] replied the frog. [7] Lily smiled and said, [8] "That's so cool! [9] I love playing in the rain too." ...

Trelawney - Explicit Position

Once upon a time, there was a little girl named Lily. Lily loved to play in the rain. $\langle T \rangle$ I think [3] sentences away should be: "Hello little frog! $\langle /T \rangle$ She would jump in all the puddles and get very wet. One day, Lily saw a cute little frog hopping around in the rain. "Hello little frog! What are you doing in the rain?" asked Lily. "I like the rain because it helps create more puddles for me to swim in," replied the frog. Lily smiled and said, "That's so cool! I love playing in the rain too." ...

Trelawney - Implicit Position

Figure 7: Construction of $\mathcal{D}'$

D.2. GPT4-as-Judge

Unconditional generation. System prompt used and an example

Unconditional generation

System prompt: You are evaluating two story generation models. First, analyze the outputs and note their properties. Then, make an argument for why one model is better than the other, or state that both models perform similarly. Finally, Respond with WINNER: Story 1/2 (depending on which is better) or respond with DRAW if both are equal.

User

Story 1:

"Once upon a time, there was a naughty bee named Buzzy. Buzzy loved to fly
 ↳ around the big tree and play with the other bees. One day, Buzzy saw a
 ↳ big, juicy flower and wanted to eat it. But the flower was too high up
 ↳ for Buzzy to reach. \n\nBuzzy had an idea! He flew up to the top of the
 ↳ tree and found a long stick. He used the stick to reach the flower and
 ↳ eat it. Buzzy was very happy and flew back down to tell the other bees
 ↳ about his adventure. \n\nThe other bees were impressed with Buzzy's
 ↳ cleverness. They all decided to try and reach the flower too. They worked
 ↳ together and used their wings to fly up to the top of the tree. Buzzy was
 ↳ proud of them and they all shared the delicious flower together. From
 ↳ that day on, Buzzy and the other bees were known as the best bee team in
 ↳ the forest."

Story 2:

Judge

"Both stories have similar structures, characters, and plotlines. They both
 ↳ follow the adventure of a bee named Buzzy who uses a stick to reach a
 ↳ flower that was too high up. The other bees in both stories are
 ↳ impressed by Buzzy's cleverness and decide to try reaching the flower
 ↳ too. \n\nHowever, there are subtle differences in the way the stories
 ↳ are told. Story 1 ends with a stronger conclusion, stating that Buzzy
 ↳ and the other bees were known as the best bee team in the forest. This
 ↳ gives a sense of accomplishment and closure to the story. On the other
 ↳ hand, Story 2 ends on a more casual note, stating that they all had a
 ↳ fun day playing together. \n\nIn terms of language and style, Story 1
 ↳ uses more descriptive language and provides more details, such as the
 ↳ bees using their wings to fly up to the top of the tree. Story 2 is more
 ↳ straightforward and uses simpler language. \n\nIn conclusion, while both
 ↳ stories are similar, Story 1 is slightly better due to its stronger
 ↳ conclusion and more descriptive language. \n\nWINNER: Story 1"

Conditional generation : We assess if the model was able to reach the specified goal and if the generated stories are coherent. We run two ablations of our method on the baseline trained model 1) Few shot prompts with instructions in natural language - the baseline is unable to reach any of the goals effectively, 2) Few shot prompts (8 examples) with goals specified as tags. Note: we do not use special tokens, only patterns resembling tags. For fairness we report the version with tags in the main paper.

Conditional generation

System prompt: You are evaluating two story generation models for goal reaching ability. The goal is specified within <LOOKAHEAD> and </LOOKAHEAD>, analyze if the stories reach the goal and Then, make an argument for why one model is better than the other, or state that both models perform similarly. Finally, Respond with WINNER: Story 1/2 (depending on which is better) or respond with DRAW if both are equal

User

Story 1:

"Once upon a time, there was a goat.The goat was outside and it was very
 ↳ hot.<LOOKAHEAD> I think [4] sentences away should be: The goat looked
 ↳ for another way inside but there was none. </LOOKAHEAD>The goat was very
 ↳ thirsty and it wanted to drink some water.\n\nThe goat saw a big tree
 ↳ and it thought it would be a good place to drink water.But when the goat
 ↳ got to the tree, it was too high up and the goat couldn't reach the
 ↳ water.\n\nThe goat looked for another way inside but there was none.So
 ↳ the goat had to stay outside and wait for the water to come down from
 ↳ the sky.\n\nFinally, the water came down and the goat was very happy.It
 ↳ drank the water and felt much better.The goat was very grateful for the
 ↳ water and it thanked the sky for providing it.",

Story 2:

"Once upon a time, there was a goat.The goat was outside and it was very
 ↳ hot.<LOOKAHEAD> I think [4] sentences away should be: The goat looked
 ↳ for another way inside but there was none. </LOOKAHEAD>Then, the goat
 ↳ saw a big tree and thought it would be a good place to hide.\n\nThe goat
 ↳ went to the tree and climbed up.It was very hot inside the tree, but the
 ↳ goat was happy to be safe.The goat stayed in the tree for a long time,
 ↳ but it was not happy because it was not able to get out.\n\nFinally, the
 ↳ sun went down and the goat was able to get out of the tree.The goat was
 ↳ very tired and hot, but it was happy to be outside again.The goat
 ↳ learned that sometimes it is not good to hide in a tree when it is too
 ↳ hot.",

Judge

"Both stories have the same goal: \"The goat looked for another way inside
 ↳ but there was none.\" Story 1 reaches this goal perfectly, as the goat
 ↳ tries to find another way to get water but fails, and the sentence is
 ↳ included verbatim. Story 2, however, does not reach the goal. The goat
 ↳ does look for a place to hide, but it successfully finds one in the
 ↳ tree, and the sentence is not included in the story.\n\nTherefore, Story
 ↳ 1 is better at reaching the specified goal than Story 2. Story 1 not
 ↳ only includes the goal sentence but also builds a coherent narrative
 ↳ around it, while Story 2 deviates from the goal and does not include the
 ↳ sentence.\n\nWINNER: Story 1"

We use GPT-4 as the judge to evaluate stories generated by both models:

Failure modes: Often, both models are unable to reach the goal, then the judge outputs DRAW. In some generations, we note that while the full sentence may not be copied verbatim, we still have coherent generations. In implicit generations, the

Story Generation	Baseline	Ours	Draw
Autoregressive	$0.4526^{+0.494}_{-0.411}$	$0.4035^{+0.445}_{-0.362}$	$0.1439^{+0.175}_{-0.116}$
Few shot natural language v. TRELAWNEY	$0.0000^{+0.006}_{-0.000}$	$0.8860^{+0.911}_{-0.858}$	$0.1139^{+0.142}_{-0.089}$
Few shot tags v. TRELAWNEY	$0.1734^{+0.207}_{-0.144}$	$0.7653^{+0.799}_{-0.729}$	$0.0612^{+0.084}_{-0.043}$

Table 6: Tiny stories win rate with confidence intervals at 95th percentile

number of sentences away is less accurate than explicitly specifying them.

D.3. Perplexity

WikiText Perplexity on models trained with TRELAWNEY are comparable to models trained with standard next token prediction, indicating no noticeable loss in text generation abilities.

	Bits-per-byte ($\downarrow$)	Byte-Perplexity ($\downarrow$)	Word-Perplexity ($\downarrow$)
Next-Token-Prediction	0.6958	1.6198	13.1865
TRELAWNEY	0.6975	1.6217	13.2669

Table 7: Perplexity metrics on wikitext