Noise Your Prompt: Prompt Noising for Conditional Generation in Continuous Diffusion Language Models

Lateral Intelligence Team*

Abstract

We revisit a standard accepted practice in the continuous diffusion language model literature of fixing conditioning prompt tokens clean during training. Rather, we make a very simple modification: also noise the conditioning prompt tokens during training. We demonstrate that under this modified training objective, we achieve better generalization in combinatorial reasoning tasks such as Sudoku and N-Queens, with significant improvements on harder variants (3.73% $\rightarrow$ 24.25% solve rate on Sudoku Hard). Moreover, our models exhibit higher generation diversity. Our method can be seen as conditioning augmentation and a generalization of classifier-free guidance training.

1 Introduction

Fundamentally, diffusion and flow models learn a score or velocity corresponding to an ODE which can be integrated to transform a base distribution to the target data distribution. Their objective can be parameterized in different ways, including the denoising parametrization $\hat{x}_1(x_t)$, where the goal is to predict a clean target data x_1 given a current noisy input x_t.

Our framework is to view this denoising learning objective from the lens of conditional prediction: given a noisy input, predict a clean target. We then see the varying levels of noise corruption of the input as a spectrum of prediction task difficulties, where the model is trained on harder (more noisy) inputs and easier (less noisy) inputs.

In practice for language modelling, conditional generation is the main paradigm: given a conditioning prompt, generate a response. Currently, conditional generation in continuous diffusion language models and flow language models utilize the following learning objective: given un-noised conditioning tokens x^c and a noised response y_t, predict the target response y_1. Likewise in sampling, the conditioning tokens are fixed and non-noised.

We improve this conditional learning objective by allowing conditional tokens to also be noised, where the model is given a noised conditional input x_t^c along with the current noised response y_t. We see this modification as extending the range of prediction tasks difficulties, where the prediction problem $(x, y_t) \rightarrow y$ is made more difficult not just by increasing the amount of noise in the current response input y_t but also by increasing the amount of noise in the conditioning input. Moreover, by noising the conditioning input, our model has a wider target that it can learn from (also needs to predict the denoised conditioning input), making learning more efficient. Varying the level of noise on the conditioning input can be seen as conditioning augmentation and a generalization of classifier-free guidance training. Empirically, we verify on various reasoning and generalization tasks, such as Sudoku and N-Queens. We find that by additionally noising the conditioning inputs, we get better generalization on these reasoning benchmarks and exhibit higher levels of generation diversity.

2 Background and Related Work

Diffusion and flow generative models. Denoising diffusion and score-based models generate data by gradually corrupting samples toward a simple base distribution and

*Correspondence: justin@lateralintelligence.org

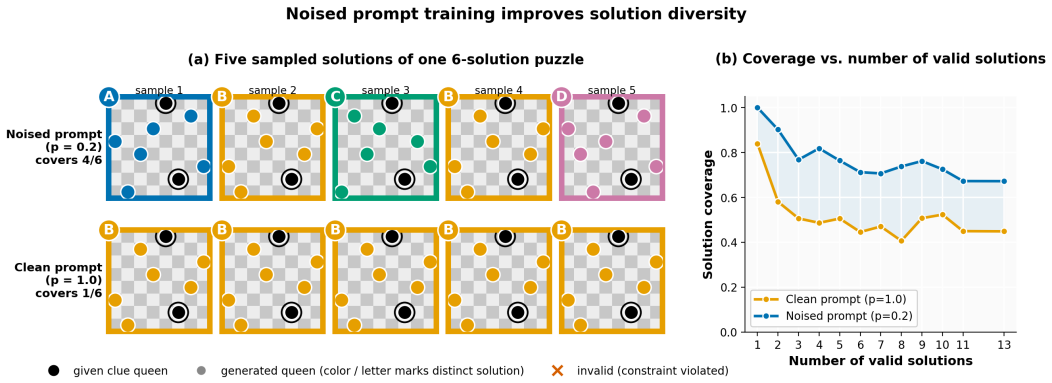

Figure 1: **Noising the conditioning tokens improves solution diversity.** *(a)* Five board generations of a single held-out N-Queens (8×8) puzzle that admits six distinct valid solutions (distinguished by color and letter). A model trained with noised conditioning tokens ($p_{\text{condclean}}$=0.2, top) spreads its generations across the solution set (4/6 distinct solutions), whereas the standard "paste-in" model that keeps conditioning tokens clean throughout training ($p_{\text{condclean}}$=1.0, bottom) mode-collapses onto a single completion. *(b)* Solution coverage as a function of the number of valid solutions admitted by the input board, over 750 held-out 10×10 puzzles with k=20 generations each. The noised-prompt model dominates the clean-prompt model at every solution count. We quantify this gap in Section 4.4.

learning to reverse this process (Ho et al., 2020; Song et al., 2021). Flow matching and stochastic interpolants (Lipman et al., 2023; Albergo & Vanden-Eijnden, 2023) give a closely related and more general view: a time-dependent interpolation between the base and data distributions defines a velocity field whose ordinary differential equation (ODE) transports noise to data, with diffusion as a special case. The learning target admits several equivalent parameterizations—predicting the injected noise, the velocity, the score, or directly the clean datapoint x_1 from a noised input x_t (x_1-prediction). We work throughout in the flow and x_1-prediction setting.

Diffusion and flow language models. Applying these models to text is complicated by the discrete nature of tokens. One line of work operates directly in discrete state spaces, defining absorbing or uniform corruption processes over tokens: D3PM (Austin et al., 2021), score-entropy discrete diffusion (SEDD) (Lou et al., 2024), masked diffusion language models (MDLM) (Sahoo et al., 2024), and large-scale masked diffusion LMs such as LLaDA (Nie et al., 2025). A second line embeds tokens into a continuous space, running continuous gaussian diffusion over those representations, and converting back to tokens: Diffusion-LM (Li et al., 2022), DiffuSeq (Gong et al., 2023), and CDCD (Dieleman et al., 2022). Recent work operates specifically on one-hot vectors on the probability simplex or on learned / pre-trained embeddings: Flow Map Language Models (Lee et al., 2026), Discrete Flow Maps (Potaptchik et al., 2026), hyperspherical flows (Deschenaux & Gulcehre, 2026), and embedded language flows (ELF) (Hu et al., 2026). Our work builds on this continuous-flow family and adopts its standard denoising (x_1-prediction) objective. By default, these models do conditional generation by holding conditioning tokens clean throughout training and moreover during sampling, where conditioning tokens are re-pasted at every sampling step (Lee et al., 2026; Hu et al., 2026). We term this default method as *paste-in* conditioning.

Diffusion forcing (Chen et al., 2024) and AR-Diffusion (Wu et al., 2023) assign per-token noise levels rather than a single shared noise level for the entire sequence. This framework motivates the notion of multiple noise levels across the sequence, where conditioning tokens may carry a different noise level than response tokens.

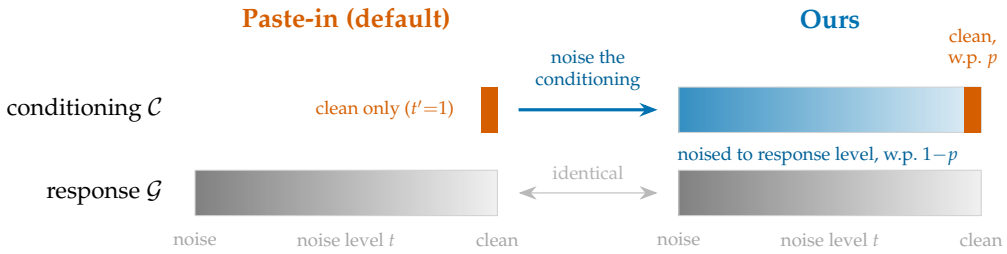

Figure 2: **Visualization of our training objective.** Standard conditional training keeps conditioning tokens at fixed clean noise level (t'=1). In contrast, we train over a range of conditioning noise levels (blue region), exposing the model to a range of conditioning levels. At generation time we follow standard practice and integrate with the conditioning tokens held clean.

Classifier-free guidance. Classifier-free guidance (Ho & Salimans, 2022) jointly trains a conditional and an unconditional model by randomly dropping the conditioning signal, then combines their predictions at sampling time to trade off fidelity against diversity. The amount of conditioning signal in the sequence is binary: it is either fully injected or fully discarded. Our method can be seen as a continuous generalization of CFG training: the amount of conditioning signal can be smoothly interpolated by the amount of noise (parametrized by time $t \in [0, 1]$), where the end points zero noise ($t = 1$) and full noise ($t = 0$) recovers the binary CFG inclusion/exclusion.

Corrupting conditioning tokens. Corrupting the conditioning information during training has appeared in several settings, though rarely for language. In image generation, cascaded diffusion models (Ho et al., 2022) apply conditioning augmentation, adding noise to the low-resolution conditioning image fed to each super-resolution stage; this improves robustness to previously produced artifacts. Classifier-free guidance (Ho & Salimans, 2022) can likewise be seen as the extreme case of corrupting the conditioning by removing it entirely. For language, corrupting the conditioning is far less explored and the evidence is mixed. Blockwise SFT for diffusion language models (Sun et al., 2025) reports that model performance degrades as the amount of noise on noisy prefixes increases, and so explicitly keeps the prefix clean.

Most directly related, prompt infilling work for masked diffusion language models (Fujinuma & Sakaguchi, 2026) unlocks prompt infilling by masking the prompt as well as the response during training, rather than masking the response alone. (Fujinuma & Sakaguchi, 2026) generate multiple prompts and select the best one, offering an alternative to prompt engineering. In contrast, our method noises conditioning tokens in continuous diffusion models during training across a continuum of corruption levels, which can be seen as expanding the range of prediction tasks. Moreover, there are no changes to inference; the inference-time conditioning tokens are kept clean and no further additional generation is necessary.

Combinatorial reasoning and generalization benchmarks. We evaluate on Sudoku and N-Queens, constraint-satisfaction tasks where solving unseen instances requires learning the underlying rules rather than memorizing input/output pairs. Both have been used to measure reasoning and moreover generalization ability in recursive and generative reasoning models (Jolicoeur-Martineau, 2025; Baek et al., 2026). In particular, N-Queens admits many valid solutions per board, which lets us measure not only accuracy but also the diversity of generated solutions, following GRAM (Baek et al., 2026).

3 Method

Our method is simple and requires minimal modification from the default training objective of flow language models. It can also be seen as a generalization of classifier free guidance

training. Simply put, with some probability $p_{\text{condclean}}$ we keep the conditioning tokens clean; else, we noise the conditioning tokens as well.

During sampling, following the default practice in diffusion/flow language model literature (Lee et al., 2026; Hu et al., 2026; Deschenaux & Gulcehre, 2026), we keep the conditioning tokens clean and paste-in clean tokens at each sampling step. To prevent a large training and inference mismatch, with some probability $p_{\text{condclean}}$ we choose to keep the conditioning tokens clean to supply as input. The model then predicts using inputs $(x_{t=1}, y_t)$, where $x_{t=1}$ are the clean conditioning tokens and y_t are the noised response tokens.

At the same time, we do not want the model to become too reliant on the conditioning tokens and possibly learn a brittle mapping $x \to y$. Thus, with the remaining probability $1 - p_{\text{condclean}}$, we noise our conditioning tokens, reducing the amount of conditioning signal the model has access to. It receives as input both noised conditioning tokens and noised response tokens (x_t, y_t). As visualized in Figure 2, by additionally noising our conditioning tokens, our model is trained under a broader curriculum of conditional prediction tasks, where the model receives both a lot of conditioning signal (making the prediction task easier) and little conditioning signal (making the prediction task harder).

Additionally, we make use of the conditioning tokens in the loss and have the model predict denoised versions of the conditioning tokens x_t. This utilizes all the tokens in the denoising objective, making the training objective more efficient. We report the effect of this in the ablation results, Section 4.5.

3.1 Setup

We work with an in-place *infilling* layout. Each example is a single sequence of length L (Sudoku: $L = 81$; N-Queens: $L = n^2$) over a small vocabulary. The model receives a sequence of per token noise levels; the model receives no formal distinction between conditioning tokens $\mathcal{C}$ and response tokens $\mathcal{G}$. We compute the loss over the full sequence.

3.2 Corruption process and varying conditioning signal

We adopt a continuous-time flow (x_1-prediction) model with per-token noise levels (as in *diffusion forcing* (Chen et al., 2024)): each position i carries its own time $\tau_i \in [0, 1]$, with $\tau_i = 1$ denoting a clean token and $\tau_i = 0$ full noise, and $x_i(\tau_i)$ is the corresponding corrupted input. The denoiser f_θ receives the noised sequence together with the per-token times and predicts the clean target $\hat{x}_1$.

A single scalar $p \equiv p_{\text{condclean}}$ controls the probability of corruption of conditioning tokens. For each example we draw a shared time $t \sim \mathcal{U}(0, 1)$ and either have *all* the conditioning tokens match the noise level of the response tokens or have *all* remain clean:

$$c \sim \text{Bernoulli}(p), \qquad \tau_i = t \;\; (i \in \mathcal{G}),$$

$$\tau_i = \begin{cases} 1 & \text{if } c = 1 \;\; (\text{clue clean}), \\ t & \text{if } c = 0 \;\; (\text{clue noised}), \end{cases} \qquad (i \in \mathcal{C}).$$

The training objective is the cross-entropy between the prediction and the original sequence over the loss region $\mathcal{L}$ (the full sequence):

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{x_1, t, \{\tau_i\}} \Big[\textstyle\sum_{i \in \mathcal{L}} \text{CE}\big(f_\theta(x(\tau))_i,\; (x_1)_i\big) \Big].$$

Setting $p = 1$ recovers the standard "paste-in" conditional objective (conditioning tokens always clean); $p = 0$ recovers a fully unconditional objective. Intermediate p interpolates between the two objectives, allowing the model to train under a range of conditioning strengths while also preventing a large training/inference gap.

3.3 Sampling

At inference we follow standard practice for conditional flow language models: the conditioning input is always clean and clamped to their clean one-hot values and re-pasted after

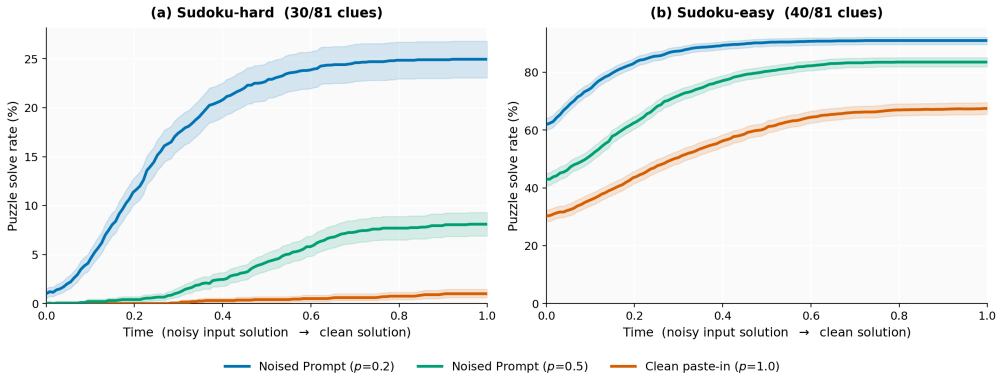

Figure 3: **Puzzle solve rate over the sampling trajectory.** Puzzle (exact match) solve rate as a function of sampling time (noisy input solution → clean solution), read from the model's x_1 prediction at each step. Means calculated over 2000 held-out puzzles with a 128-step Euler sampler; shaded bands are 95% bootstrap intervals over puzzles. Across both difficulties the noised-prompt models ($p_{\text{condclean}}$=0.2, 0.5) reach a substantially higher solve rate than the standard paste-in model ($p_{\text{condclean}}$=1.0) and also saturate their solve rate bound earlier in generation.

every solver step. We integrate with a first-order Euler sampler. Since sampling assumes conditioning tokens are kept clean, a small fraction p of training examples with clean conditioning tokens is necessary to maintain generation performance, as demonstrated in the ablations Section 4.5.

4 Experiments

4.1 Setup

Across all experiments, we use the Flow Language Model (FLM) from Lee et al. (2026). The backbone is a DiT-style transformer (Peebles & Xie, 2023) with hidden size 512, 8 blocks, 8 attention heads, conditioning dimension 128, and dropout 0.1, trained with a flow objective and x_1-prediction. We optimize with AdamW (learning rate 3×10^{-4}, $(\beta_1, \beta_2) = (0.9, 0.999)$, no weight decay), a constant schedule with 2,500 warmup steps, gradient clipping at 1.0, bf16-mixed precision, and an EMA of 0.9999. We follow FLM and use antithetic time sampling and a t-reparametrization technique that adjusts for the vocabulary size (Lee et al., 2026). Unless noted, we sample with a 128-step Euler solver. The single swept variable is $p_{\text{condclean}} \in \{0.2, 0.5, 1.0\}$ (and 0.0 where noted); all other settings are held fixed. Full hyperparameters are in Appendix B. All runs train on a single NVIDIA L40S GPU. For Sudoku, the vocabulary designates token 0 as PAD, token 1 an empty cell, and tokens 2–10 as the digits 1–9; for N-Queens, token 0 is PAD, token 1 is an empty cell, and token 2 is a queen. Following prior work (Deschenaux & Gulcehre, 2026), we train with no data augmentation.

4.2 Metrics

We report two metrics. **Accuracy** is the fraction of generated boards that are valid solutions to the puzzle (aggregated across all puzzles and samples). **Coverage** (N-Queens) is a per puzzle statistic capturing the fraction of *distinct* valid solutions recovered across $k = 20$ generated samples, all conditioned on the same input puzzle. This allows us to measure the diversity of the generations.

For Sudoku, we report 95% two-sided t-distribution confidence intervals obtained via $n = 5$ training seeds. For N-Queens, we report 95% bootstrapping confidence intervals obtained by bootstrap resampling over the evaluation puzzles (10,000 resamples; for N-Queens

Table 1: Sudoku solve rate (%) on 2,000 held-out puzzles under the conditioning clean probability $p_{\text{condclean}}$. At N=10,000, lower p (more frequently noised clues) yields higher accuracy on both difficulties; at the data-limited N=1,000 p=0.5 performs better. Brackets give 95% CIs bootstrapped over 5 training seeds, except where noted.

	Easy (40/81 clues)		Hard (30/81 clues)	
$p_{\text{condclean}}$	$N = 10{,}000$	$N = 1{,}000$	$N = 10{,}000$	$N = 1{,}000$
0.0 (uncond.)[a]	1.50 [1.00, 2.05]	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]
0.2	**88.89** [84.81, 92.97]	1.39 [0.11, 2.67]	**24.25** [20.04, 28.46]	0.01 [0.00, 0.04]
0.5	87.51 [80.93, 94.09]	**6.92** [0.86, 12.98]	14.70 [8.19, 21.21]	**0.21** [0.16, 0.26]
1.0 (paste-in)	68.49 [57.44, 79.54]	1.47 [0.00, 3.11]	3.73 [2.76, 4.70]	0.00 [0.00, 0.00]

[a] The unconditional baseline is reported from a single training seed; we report a 95% bootstrap CI. All other entries are bootstrapped over 5 seeds.

the resampling is over 750 held-out input boards, with per-board accuracy and coverage computed from that board's $k = 20$ generations).

4.3 Sudoku

Sudoku is a constraint-satisfaction puzzle on a 9×9 grid and a common benchmark for reasoning and generalization (Jolicoeur-Martineau, 2025; Baek et al., 2026). We use it precisely to measure generalization: to solve unseen puzzles a model cannot simply memorize a mapping from $x \to y$, it must implicitly learn and respect the constraints of the game.

We train on Sudoku with the in-place infill layout: a full 9×9 board flattened row-wise (length 81), with the loss taken over all board tokens. Puzzles are generated following Deschenaux & Gulcehre (2026), and difficulty is controlled by the number of revealed clues—40 of 81 cells given for *easy* and 30 of 81 for *hard*. We sweep $p_{\text{condclean}}$ at two training-set sizes, $N = 10{,}000$ and $N = 1{,}000$ puzzles and evaluate accuracy on 2,000 held-out puzzles with a 128-step sampler (Table 1).

The default setup of pasting-in clean conditioning tokens during training performs notably worse than our method of noising conditioning tokens across all Sudoku settings. On Sudoku easy, p=0.2 reaches 88.89% versus 68.49% for the standard paste-in setup; on hard, 24.25% versus 3.73%. Under the data-limited regime $N = 1{,}000$, p=0.5 performs best; the default paste-in strategy and p=0.2 perform similarly, with no statistically significant difference in performance.

The fully-unconditional setup ($p = 0.0$) exhibits poor generation, demonstrating that the model still needs to see some conditioning tokens as clean.

4.4 N-Queens

N-Queens is a puzzle where n mutually non-attacking queens must be placed on a $n \times n$ board and is likewise used to evaluate reasoning and generalization of models (Baek et al., 2026). In addition to being combinatorial in nature, N-Queens admits multiple valid solutions for a given starting board. This allows us to measure the diversity of the generated solutions. Following GRAM (Baek et al., 2026), we track coverage of possible solutions and report the fraction of unique solutions generated out of the total number of possible solutions corresponding to that board. We evaluate on held out initial board states and generate 20 samples per input board.

We consider two scales, $n = 8$ (length 64) and $n = 10$ (length 100). We subsample the data such that puzzles of varying difficulty (as defined by the number of initial queens) are

Table 2: N-Queens infill accuracy and coverage (%), final checkpoint, 750 puzzles $\times$ 20 samples. Both accuracy and coverage are monotonic in $p_{\text{condclean}}$ where p=0.2 dominates throughout.

	$n = 8$ (length 64)		$n = 10$ (length 100)	
$p_{\text{condclean}}$	Accuracy	Coverage	Accuracy	Coverage
0.2	**99.16**	**92.97**	**97.90**	**73.93**
	[98.61, 99.60]	[91.75, 94.16]	[97.37, 98.37]	[72.54, 75.36]
0.5	98.97	90.88	94.23	60.68
	[98.48, 99.39]	[89.51, 92.22]	[93.25, 95.14]	[59.12, 62.28]
1.0 (paste-in)	97.67	88.07	84.80	49.89
	[96.73, 98.51]	[86.46, 89.63]	[83.09, 86.42]	[48.21, 51.56]

equal in number. For $n = 8$ this yields 1,758 training / 174 validation puzzles; for $n = 10$ we have 5,717 training / 557 validation puzzles. We also hold-out a seperate evaluation partition of 750 input boards for both $n = 8$ and $n = 10$. Training and evaluation splits are disjoint by input board. We train at global batch size 256 for 100k steps and evaluate the final checkpoint on 750 held-out puzzles with 20 generations each (Table 2).

At $n = 8$ accuracy is high throughout all p; whereas at $n = 10$ the noised conditioning tokens at ($p = 0.2$) performs substantially better over the default paste-in setup.

Coverage is a more salient metric: for both $n = 8$ and $n = 10$, training on just clean conditioning tokens leads to lower diversity of generations, prominently demonstrated at $n = 10$ (49.89 vs 73.93%)

4.5 Ablations

We consider the following three ablations to illustrate the design choice of our noising scheme: (i) whether the conditioning tokens are noised to the same time as the response or to an independent time; (ii) whether the loss is computed over the full sequence or only the response tokens; and (iii) the whether the model needs to see clean conditioning tokens at all (fully unconditional training). All numbers are calculated with 95% bootstrap CIs (10,000 resamples; Sudoku n=2,000, N-Queens 750 input boards).

Separate conditioning time. The current objective noises conditioning tokens to the same sampled noise level (parameterized by time) as the response tokens. We consider drawing an independent time $t' \sim \mathcal{U}(0, 1)$ for the conditioning tokens, so the conditioning and response are corrupted to decoupled noise levels.

On Sudoku easy (N=10,000) it reaches 90.45%, above the best single-time setting in Table 1 (87.55% at p=0.2), while on Sudoku-hard it performs worse at 21.80%. On N-Queens it does not beat frequent same time noising (Table 2).

Loss calculated on full sequence vs response only. We ablate a response-tokens only loss objective that excludes the conditioning tokens predictions, sweeping $p_{\text{condclean}}$ on Sudoku-easy (N=10,000). By incorporating only the response tokens in the loss, the model performs worse, validating our position that a full sequence objective is a more effective training objective.

Notably, the noised prompt training still significantly outperforms paste-in clean prompt training (83.80% at p=0.2 vs 36.95% at p=1.0), demonstrating that the performance gains mainly come from noising, rather than a full sequence loss objective.

Only noised conditioning tokens ($p_{\text{condclean}}$=0). With conditioning tokens always noised, the model learns the unconditional distribution but is unable to utilize the given clean conditioning information at inference time. On Sudoku hard (N=10,000) accuracy collapses

Table 3: Ablations (accuracy / coverage %, with 95% bootstrap CIs). *Top:* separate conditioning-time variant (independent clue time t'). *Middle:* response-only loss region on Sudoku-easy (N=10,000), swept over $p_{\text{condclean}}$. *Bottom:* Fully unconditional generation.

Ablation	Setting	Accuracy	Coverage
Separate cond. time	Sudoku easy	90.45 [89.15, 91.70]	—
	Sudoku hard	21.80 [20.00, 23.60]	—
	N-Queens n=8	97.31 [96.32, 98.18]	89.52 [88.06, 91.00]
	N-Queens n=10	87.41 [85.83, 88.91]	51.40 [49.74, 53.10]
Response-only loss (Sudoku easy)	p=0.0	2.60 [1.95, 3.30]	—
	p=0.2	83.80 [82.20, 85.40]	—
	p=0.5	46.50 [44.35, 48.65]	—
	p=1.0	36.95 [34.90, 39.15]	—
Unconditional (p=0)	Sudoku easy	1.50 [1.00, 2.05]	—
	Sudoku hard	0.00 [0.00, 0.00]	—

to 0%; and on Sudoku easy (N=10,000) accuracy is 1.50%. This confirms that some exposure to clean conditioning tokens is necessary.

5 Analysis and Discussion

Why does noising the conditioning help? We posit views consistent with the results above. First, noising the conditioning tokens leads to a wider set of training inputs (seen as data augmentation) and also makes use of the conditioning token predictions in the loss. Second, by decreasing the amount of signal in conditioning tokens, the model becomes less reliant on predicting responses mainly based on the condition, leading to a greater variety of generations, as exhibited by the mode coverage results (always-clean training ($p = 1$) collapses toward fewer solutions). However, we find that actually noising the conditioning tokens is critical (Appendix A.1). Finally, incorporating both noisy conditional tokens and non-noisy conditioning tokens can be seen as expanding the learning objective: $p < 1$ mixes an easier prediction task (given fully clean condition signal) with a harder (given noised conditioning signal) prediction task.

Relation to classifier-free guidance. Classifier-free guidance trains with a binary inclusion/exclusion of conditioning information. Our model can be seen as a generalization of CFG training, where it trains with a spectrum of conditioning information noise levels. The endpoints of full noise ($t = 0$) and no noise ($t = 1$) recovers the CFG inclusion/exclusion. Currently we only consider fully clean conditioning tokens at inference time; combining predictions under fully clean conditioning tokens and partially noised conditioning tokens during sampling may be a natural next direction.

Limitations. We train on relatively small models and trained on small datasets. Future work would include evaluating on much larger models and consider scaling behavior. For some experiments, we report bootstrap CIs which control for evaluation sampling variance, but not for training seed variance. We evaluate on combinatorial reasoning tasks such as Sudoku and N-Queens. While we have not run sufficient experiments on natural language reasoning datasets due to compute restrictions, this is a direction we hope to see.

6 Conclusion

We revisited an accepted default practice of keeping conditioning tokens clean during training of continuous diffusion language models and showed that simply noising the conditioning tokens as well during training improves conditional generation performance and generation diversity. The method is a trivial engineering change to the training objec-

tive and simple to implement. Future work includes scaling analysis with larger models, using language dataset benchmarks, and incorporating noised conditioning tokens during sampling-time.

7 Acknowledgements

We thank Julian Quevado for his extended encouragement and support. We thank the broader diffusion language model community for their foundational work and ongoing contributions to this area of research.

References

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2023. URL https://arxiv.org/abs/2209.15571.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006*, 2021. URL https://arxiv.org/abs/2107.03006.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026. URL https://arxiv.org/abs/2605.19376.

Boyuan Chen, Diego Martí Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *arXiv preprint arXiv:2407.01392*, 2024. URL https://arxiv.org/abs/2407.01392.

Justin Deschenaux and Caglar Gulcehre. Language modeling with hyperspherical flows. *arXiv preprint arXiv:2605.11125*, 2026. URL https://arxiv.org/abs/2605.11125.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022. URL https://arxiv.org/abs/2211.15089.

Yoshinari Fujinuma and Keisuke Sakaguchi. Unlocking prompt infilling capability for diffusion language models. *arXiv preprint arXiv:2604.03677*, 2026. URL https://arxiv.org/abs/2604.03677.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. DiffuSeq: Sequence to sequence text generation with diffusion models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.08933.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. URL https://arxiv.org/abs/2207.12598.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020. URL https://arxiv.org/abs/2006.11239.

Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *arXiv preprint arXiv:2106.15282*, 2022. URL https://arxiv.org/abs/2106.15282.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. ELF: Embedded language flows. *arXiv preprint arXiv:2605.10938*, 2026. URL https://arxiv.org/abs/2605.10938.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025. URL https://arxiv.org/abs/2510.04871.

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026. URL https://arxiv.org/abs/2602.16813.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2205.14217.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2023. URL https://arxiv.org/abs/2210.02747.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2024. URL https://arxiv.org/abs/2310.16834.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025. URL https://arxiv.org/abs/2502.09992.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, 2023.

Peter Potaptchik, Jason Yim, Adhi Saravanan, Peter Holderrieth, Eric Vanden-Eijnden, and Michael S. Albergo. Discrete flow maps. *arXiv preprint arXiv:2604.09784*, 2026. URL https://arxiv.org/abs/2604.09784.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*, 2024. URL https://arxiv.org/abs/2406.07524.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2021. URL https://arxiv.org/abs/2011.13456.

Bowen Sun et al. Blockwise SFT for diffusion language models: Reconciling bidirectional attention and autoregressive decoding. *arXiv preprint arXiv:2508.19529*, 2025. URL https://arxiv.org/abs/2508.19529.

Tong Wu, Zhihao Fan, Xiao Liu, Yeyun Gong, Yelong Shen, Jian Jiao, Hai-Tao Zheng, Juntao Li, Zhongyu Wei, Jian Guo, Nan Duan, and Weizhu Chen. AR-Diffusion: Auto-regressive diffusion model for text generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.09515.

A Further Ablations

A.1 Noising conditioning time but keeping conditioning data clean

Table 4: Effect of noising the conditioning *time* while keeping the conditioning *data* clean. p is the probability that a conditioning tokens's time value stays clean, i.e remains $t = 1$. 95% bootstrap CIs reported.

	Conditioning-time clean prob. p		
Difficulty	0.2	0.5	1.0
Easy	66.95 [64.95, 69.00]	69.55 [67.55, 71.60]	68.60 [66.60, 70.60]
Hard	4.80 [3.90, 5.75]	5.50 [4.55, 6.50]	3.75 [2.95, 4.60]

We ablate whether the improvement in model performance under noised conditioning prompts stems from the increase in variation of conditioning inputs, hinting at a data augmentation effect, (i.e model trains on a range of conditioning tokens x_t, not just clean tokens x_1) or due to its decrease in reliance on the conditioning prompts for prediction, hinting at regularization effect. We only noise the time values of the conditioning tokens that the model takes as input, but keep the actual conditioning tokens input the same, i.e one-hot vectors. Admittedly, we see a confounding effect where the model can learn to identify one-hot vectors as clean, bypassing the supplied noised conditioning token time values. In Table 4, we see that when the model is "told" via the noised time values that the conditioning prompt is noisy, it performs simillary on Sudoku Easy and nominally better (5.50% vs 3.75%) on Sudoku Hard. For significant model improvement, it is necessary to actually noise the conditioning prompt tokens.

B Full Hyperparameters

Table 5 lists the shared training configuration.

Table 5: Shared hyperparameters

Group	Setting	Value
Model	hidden size / blocks / heads	512 / 8 / 8
	cond. dim / dropout	128 / 0.1
	board length	81 (Sudoku), 64/100 (N-Queens n=8/10)
Algorithm	pred. type	x_1
	loss type	flow
	$p_{\text{condclean}}$	$\{0.2, 0.5, 1.0\}$ (swept)
Training	max steps	$100k$ (N-Queens), 40k (Sudoku)
	global batch size	256
	precision / EMA	bf16-mixed / 0.9999
	grad clip	1.0
Optimizer	class	AdamW
	lr / (β_1, β_2)	3×10^{-4} / $(0.9, 0.999)$
	warmup	2,500 steps (constant after)
Sampling	solver	Euler
	steps	128
Hardware	accelerator / GPU	cuda, 1 device / NVIDIA L40S