SELF-AWARE MARKOV MODELS FOR DISCRETE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard masked discrete diffusion models face limitations in reasoning tasks due to their inability to correct their own mistakes on the masking path. Since they rely on a fixed number of denoising steps, they are unable to adjust their computation to the complexity of a given problem. To address these limitations, we introduce a method based on learning a Markov transition kernel that is trained on its own outputs. This design enables tokens to be remasked, allowing the model to correct its previous mistakes. Furthermore, we do not need a fixed time schedule but use a trained stopping criterion. This allows for adaptation of the number of function evaluations to the difficulty of the reasoning problem. Our adaptation adds two lightweight prediction heads, enabling reuse and fine-tuning of existing pretrained models. On the Sudoku-Extreme dataset we clearly outperform other flow based methods with a validity of 95%. For the Countdown-4 we only need in average of 10 steps to solve almost 96% of them correctly, while many problems can be solved already in 2 steps.

1 INTRODUCTION

Chain-of-thought reasoning Wei et al. (2022) has become standard in autoregressive LLMs, but relies on sequential commitment, i.e. once a token is generated, possible errors propagate further. Discrete diffusion has been shown to be a scalable method Nie et al. (2025); Gat et al. (2024); Austin et al. (2023); Campbell et al. (2024), and offer offers a natural alternative, rather than committing sequentially, the model treats its output as a revised draft, where tokens can be sampled, reconsidered, and replaced. Although this flexibility could enable errors to be corrected rather than locked in, this potential remains largely unrealized so far. In this paper, we are interested in reasoning tasks, where instead of generation problems, answers are uniquely determined, making error correction essential rather than optional. Typical examples are Sudoku and Countdown treated in our numerical part. For these problems, current discrete flow matching training exhibits two key limitations, namely limited self-correction and non-adaptivity to the problem's complexity:

i) **Self-Correction.** Under the standard masking learning process, a partially noised state contains either mask tokens or correct target tokens, but never incorrect ones. The model therefore learns to denoise only from idealized states. However, at inference, the model relies on states it has generated itself, which may contain errors. Once an incorrect token is placed, it persists, since standard samplers do not revisit unmasked positions. Consequently, subsequent predictions conditioned on a state, except for any seen during training, become unreliable.

ii) **Integration of Problem Difficulty.** Standard inference uses a predetermined number of denoising steps. Therefore, instances that are easy for the model and could be solved quickly, still require the full budget, wasting computation and risking unnecessary errors from repeated resampling. On the other hand, hard instances that require more refinement receive no additional resources. In summary, without a mechanism to assess progress, the model cannot adapt its computation to the problem at hand.

Existing approaches. Several strategies have been proposed to improve reasoning in discrete diffusion. Inference-time methods such as remasking Wang et al. (2025b) and adaptive token ordering

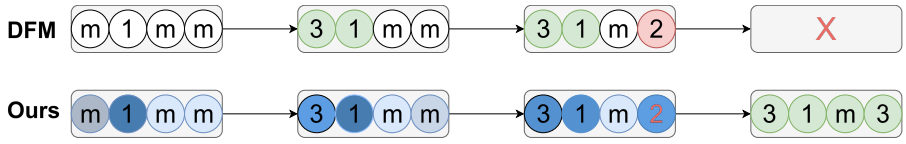

Figure 1: Inference trajectories. **DFM** Gat et al. (2024): once the sampler leaves the masking path (red, incorrect token), it fails to recover and does not converge to the target solution (X). **Ours**: trained on model-induced off-path states, the model detects the mistake and corrects it, recovering the correct final sequence (green); color intensity indicates the model's confidence.

Kim et al. (2025b) modify the sampling procedure. The procedure enables error correction without retraining, but cannot change what the model has learned. Reinforcement learning methods Zhao et al. (2025); Tang & Zhao (2025) adapt policy gradient techniques to discrete diffusion, using reward signals to improve reasoning. This is a distinct paradigm from supervised training and introduces additional complexity such as reward design and training instability. Training-time loss modifications reweight tokens by difficulty Ye et al. (2024a) or add auxiliary heads predicting per-token quality Kim et al. (2025a). It improves learning, but does not directly expose the model to states containing its own errors. GIDD of von Rütte et al. (2025b) generalizes the noise process to include uniform noise alongside masking, so that training states may contain random incorrect tokens. This exposes the model to *some* errors, but not to its own systematic mistakes. Finally, autoregressive models, also called next token-prediction are based on $p(x) = p(x_0) \prod_i p(x_i|x_{k<i})$ have many advantages in training, like KV-Caching Pope et al. (2022), and training on the whole sequence in parallel. However, they have the disadvantage that they can not generate in parallel and fix their own mistakes.

Our approach. Motivated by discrete flow matching, we propose a training framework for reasoning tasks based on a simple principle: train the model on states it actually produces, with a mechanism to abstain when uncertain, in Figure 1 this is visualized . This leads to two interdependent components. First, we learn a *Markov transition kernel* that separates *what* to predict from *when* to commit: at each position, a learned confidence score gates whether the model outputs a concrete token or remains masked. Second, we train on *on-policy states* generated by applying this mixed kernel, exposing the model to configurations containing its own errors. The training objective encourages both accuracy (predict the correct token) and safety (do not commit when uncertain). An auxiliary head estimates progress toward completion, enabling early stopping when the model is confident it has finished.

The resulting model is no longer a generative model in the probabilistic sense, since we do not maintain a valid probability path from noise to data, except when we have an unique pairing between clues and solution. Our method is tailored to tasks where correctness matters more than distributional fidelity. In our model, discrete flow matching inspires a iterative refinement mechanism and optimize directly for task performance.

Contributions. Based on the identification of the above two limitations of standard discrete flow matching for reasoning tasks, namely that the model never trains on states containing its own errors, and that computation does not adapt to instance difficulty, we propose

- a novel method combining a mixed prediction-commitment kernel with on-policy training, addressing both limitations within a supervised learning framework. First, we train on mistakes produced by the model, allowing it to self-correct. Second, we introduce a stopping time, allowing the model to spend more effort on harder problems. Finally, to solve easy problems faster, we introduce a confidence criterion that lets the model fill in easy tokens first, while handling incorrect tokens by re-masking or correcting them.
- We demonstrate improved reasoning performance on Sudoku-Extreme and Countdown compared to standard training and existing methods.

2 SELF-CORRECTING, DIFFICULTY-AWARE MARKOV MODEL

We want to model discrete data $\mathcal{S} = \mathcal{T}^d$, where $\mathcal{T} = \{T_1, \dots, T_N\} \cup \{m\}$ is a set of tokens and m is a special mask token. States (i.e., samples) from $\mathcal{S}$ are denoted by $x = (x^1, \dots, x^d)$ with $x^i \in \mathcal{T}$. By $\Delta_{\mathcal{S}}$, we denote the probability simplex with vertices $\mathcal{S}$. In the following, we restrict ourselves to the reasoning setting, where there is a unique pairing between clues and solutions. We denote the clue distribution by X_0, the solution distribution by X_1, and their pairing by π. Note that $(x_0, x_1) \sim \pi$, if x_1 is the solution of x_0. For example, in Sudoku, we have $\mathcal{T} = \{1, \dots, 9\} \cup \{m\}$ and $d = 81$. The clues are the given numbers at a certain position, while the remaining fields are filled with mask tokens.

Discrete-Time Markov Chain A homogeneos discrete-time Markov chain $(Y_n)_n$ of random variables $Y_n \in \mathcal{S}$ is characterized by its transition kernel P (transition matrix) with entries

$$P(x|y) := \mathbb{P}(Y_n = x \mid Y_{n-1} = y).$$

While most existing methods fix a forward (noising) kernel and learn a backward (denoising) kernel, see, e.g. Austin et al. (2023), we instead learn the forward kernel directly. Our goal is that, for sufficiently large n we have $P^n \delta_{x_0} \approx \delta_{x_1}$ for all $(x_0, x_1) \sim \pi$. Since we want to find a solution that matches the clue, we force our Markov kernel to leave the clues unchanged. Consequently our Markov kernel depends on x_0, which encodes the clues. For readability we omit this dependence from the notation and write P instead of P^{x_0}.

The Markov kernel should be easy to model and learn, concentrate on the correct solution, and enable self-correction and adaptive computation based on problem difficulty. To this end, we model a *conditional target distribution* $p_1^\theta(\cdot \mid y)$, an *adaptive confidence* $c^\theta(y)$ (see 2.1) that decides whether to keep a token masked, unmasked, or to re-mask it, and a *stopping time* $\tau^\theta(y)$ (see 2.2). We propose to learn a network given by

$$T_\theta : \mathcal{S} \to \Delta_{\mathcal{S}} \times [0,1]^d \times [0,1], \quad y \mapsto (p_1^\theta(\cdot \mid y), c^\theta(y), \tau^\theta(y)), \tag{1}$$

where we keep the standard assumption that we learn a product distribution of the form

$$p_1^\theta(\cdot \mid y) \coloneqq \prod_{i=1}^d p_1^{\theta,i}(x^i \mid y).$$

Based on our network, we define a Markov kernel P^θ by

$$P^\theta(x \mid y) \coloneqq \begin{cases} \delta_y(x) & \text{if } \tau^\theta(y) \geq 1 - \varepsilon, \\ \prod_{i=1}^d \Big((1 - c^{\theta,i}(y))\, \delta_m(x^i) + c^{\theta,i}(y)\, p_1^{\theta,i}(x^i \mid y) \Big) & \text{otherwise.} \end{cases} \tag{2}$$

In practice we fix the clues, i.e. we force $P^\theta(x \mid y) = 0$ if the clues of x do not match the clues of x_0. If the confidence $c^{\theta,i}(y)$ is large, the probability of the token x^i follows $p_1^{\theta,i}(x^i \mid y)$. If the confidence is small, the model may keep the position masked or re-mask it. We reapply the kernel until the stopping time is hit.

Inference. Let $\mu_0^\theta = P_{X_0}$ be the law of X_0. For inference, we start in $x_0 \sim X_0$ admitting a solution, and iteratively update the chain as

$$x_{n+1} \sim P^\theta(\cdot \mid x_n), \quad \mu_{n+1}^\theta = \mu_n^\theta P^\theta \coloneqq \sum_{y \in \mathcal{S}} P^\theta(\cdot \mid y) \mu_n^\theta(y). \tag{3}$$

Note that μ_n^θ depends on x_0, since P^θ does as well.

Training. We train the network such that the stationary distribution of P^θ approximates δ_{x_1}.

Training on the path distribution $(\mu_n^\theta)_n$ is expensive, since we can only obtain it by unrolling the network, i.e., performing a fixed number of forward steps in equation 3. Instead we use ideas from discrete flow matching, see Appendix C. We use the masking path[1]

$$p_{t|0,1}^i(\cdot \mid x_0, x_1) := (1 - \kappa_t)\, \delta_m^i(\cdot) + \kappa_t\, \delta_{x_1}^i(\cdot), \tag{4}$$

[1]For a random variable $X_t \in \mathcal{S}$, $t \in [0,1]$ with probability $p_{X_t}(x) = \mathbb{P}(X_t = x)$, we just write $p_t(x)$, and accordingly $p_{t|0,1}(x|x_0, x_1) = p_{X_t|(X_0,X_1)=(x_0,x_1)}(x)$.

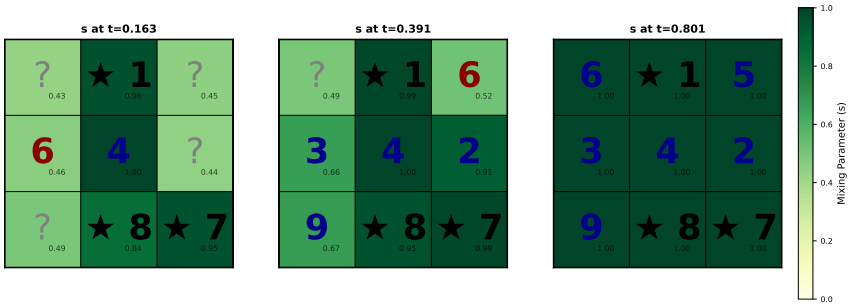

Figure 2: Inference on a 3×3 Sudoku subgrid from a complete puzzle. We show the state after $k = 1, 2, 3$ refinement steps; the predicted progress τ_θ is displayed above each panel. Cell shading encodes confidence (mixing score c_θ); clue cells are marked with $*$. Correct predictions are blue and incorrect ones red.

with a schedule function $\kappa_t : [0,1] \to [0,1]$, for example $\kappa_t = t$. We approximate $(\mu_n^\theta)_n$ by performing one Markov-kernel step on $p_{t|0,1}$, resulting in

$$\nu_t^\theta(\cdot \mid x_0, x_1) \coloneqq p_{t|0,1}(\cdot \mid x_0, x_1)\, P^\theta.$$

Allowing the network in equation 1 to learn all components, we propose to minimize the following loss, where sg denotes stop grad:

$$\begin{aligned}\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\ (x_0,x_1)\sim\pi,\ z\sim\nu_t^{\mathrm{sg}(\theta)}(\cdot|x_0,x_1)}\Big[&- \sum_{i=1}^{d} \log\left(c^{i,\theta}(z)\, p_1^{\theta,i}(x_1^i \mid z)\right) \\ &- \frac{1}{1-c_\theta^i(z)} \log\Big(1 - c^{\theta,i}(z) \sum_{v\in\mathcal{T}\setminus\{m,x_1^i\}} p_1^{\theta,i}(v \mid z)\Big) \\ &+ \big|\tau(z|x_1,x_0) - \tau_\theta(z)\big|\Big].\end{aligned} \tag{5}$$

In the following subsection, we explain our choice of the loss.

2.1 Confidence for Self-Correction

Ideally, the model decides for itself whether to unmask a token, keep it unmasked, or ultimately remask it. To learn this behaviour, the model must encounter states z^i that are neither correct nor masked, and learn the best strategy in such situations. One could introduce uniformly random mistakes into the interpolation path, as in GIDD von Rütte et al. (2025a), but this would waste training capacity on errors that the model would not have made anyway. Diffusion-Chain-of-Thought Ye et al. (2024b) instead uses, for 5% of the training states, a solution x_1 obtained by running the reverse process.

Instead, we introduce model-dependent errors by taking one forward step with our Markov kernel, which yields ν_t^θ. On states $z \sim \nu_t^\theta$, the model can learn to estimate its own reliability based on its predictions $p_1^\theta(\cdot \mid z)$. Using our kernel equation 2, the model can progress faster when it is confident, this is visualized in Figure 2. This contrasts with most diffusion models, which fix the number of inference steps. To obtain this behaviour, we maximize the probability of being correct via the term $-\sum_{i=1}^d \log\big(c^{i,\theta}(z)\, p_1^{\theta,i}(x_1^i \mid z)\big)$, while penalizing incorrect unmaskings $\frac{1}{1-c_\theta^i(z)} \log\Big(1 - c^{\theta,i}(z) \sum_{v\in\mathcal{T}\setminus\{m,x_1^i\}} p_1^{\theta,i}(v \mid z)\Big)$ in equation 5.

2.2 Difficulty-Aware Computation Time

Most discrete diffusion models use a fixed time schedule. The model must converge within a predetermined step budget. Instead, we want the model to spend more steps on harder instances.

We introduce a stopping time on an appropriate filtered probability space, which serves two purposes. First, it provides a valid stopping criterion. Second, it can be used to determine an adaptive step size. To this end, we let the network also predict its own time, defined by

$$\tau(x_t \mid x_1, x_0) = \frac{\big|\{x_t^i \mid \text{correct and not a clue}\}\big|}{\big|\{x_t^i \mid \text{not a clue}\}\big|}. \tag{6}$$

Remark 2.1. With regard to the flow-matching setting in Appendix C, we see from equation 18 that

$$\mathbb{E}_{x\sim p_{t|0,1}}\big[\tau(x \mid x_0, x_1)\big] = \sum_{x\,:\,x^i\in\{x_1^i,m\}} \tau(x \mid x_0, x_1) \prod_{i=1}^{d} \Big((1-\kappa_t)\,\delta_m(x^i) + \kappa_t\,\delta_{x_1^i}(x^i)\Big) = \kappa_t, \tag{7}$$

i.e., for $\kappa_t = t$, the expectation of $\tau(\cdot \mid x_0, x_1)$ equals the path time.

2.3 Inference Strategies

The method converges substantially faster (see Section 3). As a result, we can afford to run multiple sampling instances in parallel and aggregate them into a single prediction: for each position, we take an $\arg\max$ over the empirical token frequencies (equivalently, the averaged categorical probabilities) to obtain the most likely token at that location.

3 Experiments

All methods share the same Transformer architecture (8 blocks, 8 attention heads, rotary position embeddings) trained with AdamW and a polynomial masking schedule. Architecture dimensions, time-conditioning, and training hyperparameters differ between tasks and are detailed in Section A.

3.1 Sudoku

Sudoku is a standard logic puzzle played on a 9×9 grid, partitioned into nine 3×3 subgrids. Some cells are pre-filled as *clues*, and the goal is to fill the remaining cells with digits 1–9 such that each row, each column, and each 3×3 subgrid contains every digit exactly once.

Table 1: Sudoku solving accuracy (%). All methods except CTTT use the same backbone architecture.

Method	Sudoku Extreme	Kaggle Unfiltered
DFM (Euler)	31.6	44.5
DFM + Top-K	68.8	89.2
DFM + Top-K Margin	68.5	88.5
DFM + ReMDM	51.3	71.9
CTTT	–	98.9
Ours	**95.2**	**99.9**

Setup. We evaluate on two 9×9 Sudoku datasets of varying difficulty: **Sudoku-Extreme** Wang et al. (2025a), a collection of 3,104,157 challenging puzzles that typically require substantially more search/backtracking (i.e., "guesses") for classical solvers, and **Kaggle Unfiltered** Radcliffe (2020), a large-scale dataset of mixed difficulty with over one million puzzles (see Section A.4 for details). All models are trained on Sudoku-Extreme: each input x_0 consists of the given clues and mask tokens for all remaining cells. We train our method for 3.8 million steps, while the baseline is trained for 10 million steps since its performance continued to improve over the longer schedule. We evaluate on 1,000 held-out Sudoku-Extreme puzzles and additionally report generalization on the unseen Kaggle Unfiltered dataset.

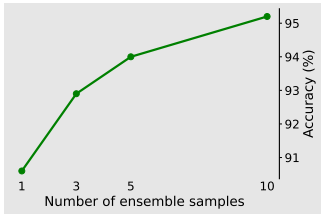

Figure 3: Sudoku accuracy vs. ensemble size.

Baselines. We compare the following methods, all using the same pretrained backbone unless noted otherwise. The baseline model is trained on the masking path equation 4 and the cross entropy loss on $p_{1|t}$, see equation 34. *DFM* Gat et al. (2024): standard discrete flow matching with Euler

sampling. For inference methods we are using *Top-K* and *Top-K Margin* Kim et al. (2025b): adaptive unmasking order based on model confidence; see Section B.1. *ReMDM* Wang et al. (2025b): inference-time remasking allowing previously decoded tokens to be reconsidered; see Section B.2. *GIDD* von Rütte et al. (2025b): pretraining with a generalized noise process mixing mask, uniform, and data tokens; see Section B.3. CTTT Giannoulis et al. (2026) is trained on its own randomly generated Sudoku dataset and evaluated on Kaggle Unfiltered. We report their published result.

Ours. Table 1 reports accuracy (the fraction of fully solved puzzles). We use 10 ensemble samples for inference (see Section 2.3), which yields a validity of 95.2%, and Table 2 reports the number of denoising steps. The Kaggle Unfiltered dataset is substantially easier then Sudoku Extreme: we solve all but one puzzle correctly. In Figure 3, we compare accuracy as a function of the ensemble size. Accuracy improves slightly with more ensemble samples, while the average number of steps increases sublinearly, from 62 to 146.

Table 2: Average sampling steps vs. ensemble size.

# Ensemble	Avg. steps
1	64.2
3	105.1
5	123.5
10	146

3.2 COUNTDOWN

Countdown Wikipedia contributors (2026) is a numerical reasoning puzzle in which one is given a multiset of integers and a target value, and must produce a valid arithmetic expression using $+, -, \times, /$ (with exact integer division) that evaluates to the target while using each number at most once. We focus on the *Countdown-4* setting (four input numbers) and design our task and evaluation *in the spirit of* Ye et al. Ye et al. (2024a), while making several implementation choices and constraints that differ from their exact setup. An example would be the numbers $86, 28, 13, 31$ and the target 96 which have to be combined as follows, $86 + 28 = 114$, $31 - 13 = 18$ and $114 - 18 = 96$.

Table 3: Countdown-4 (CD4). Baselines reproduced from Ye et al. Ye et al. (2025).

Method	Params	CD4 (%)
Autoregressive		
GPT-2 Scratch	85M	45.8
GPT-2 Scratch	303M	41.3
Stream-of-Search	250M	54.2
LLaMA	7B	41.1
LLaMA	13B	51.1
Diffusion		
VDM	85M	73.4
D3PM	85M	83.1
RDM	85M	87.0
MGDM	85M	91.5
Ours	34M	**95.9**
Small	11M	**92.1**
DFM	34M	87.5

Note: Non-ours numbers are taken from Ye et al. Ye et al. (2025) (Table 1).

Setup. We train on 500k examples generated following Ye et al. (2025); Gandhi et al. (2024), with target numbers ranging from 10 to 99. In contrast to Ye et al. (2024a), we employ a base-$10{,}000$ tokenizer in which each integer from 0 to $9{,}999$ is represented as a single token, rather than encoding numbers digit by digit. All comparisons are taken from Ye et al. (2024a), which reports results for the diffusion models VDM Kingma et al. (2023), D3PM Austin et al. (2023), RDM Zheng et al. (2024), as well as their method MGDM Ye et al. (2025). The DFM baseline is implemented by us, and we train both models for 500k steps.

Baseline. Our baseline performs comparably to the baseline reported in Ye et al. (2024a) for MGDM. We use 100 Euler steps in the discrete flow-matching solver Gat et al. (2024).

Our Method. Using an ensemble of 5 models, our method achieves 95.9% accuracy, substantially outperforming the DFM baseline (87.5%) and exceeding the best reported non-ours validity (91.5%). We also train a smaller variant with 11M parameters for 500k steps, which reaches 92.1%, still above MGDM despite using a much smaller network. With 5-model ensembling, the average number of steps is 24.7. Using a single model, we obtain 93.8% accuracy with only 7.2 average steps.

4 Conclusion

We propose a lightweight adaptation of discrete diffusion models for reasoning tasks. Our approach learns a Markov transition kernel on model-generated states, enabling self-correction by allowing tokens to be unmasked, kept, or re-masked during inference. Two additional heads provide the required control: a confidence head that guides corrective transitions and a learned stopping head that allocates a problem-dependent inference budget. Experiments on Sudoku and Countdown demonstrate that this adaptive, self-correcting inference substantially improves validity while often requiring only a small number of steps.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023. URL https://arxiv.org/abs/2107.03006.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design, 2024. URL https://arxiv.org/abs/2402.04997.

R. Duong, J. Chemseddine, P. Friz, and G. Steidl. Telegrapher's generative model via Kac flows. *arXiv preprint arXiv::2506.20641*, 2025.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024. URL https://arxiv.org/abs/2404.03683.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *arXiv preprint arXiv:2407.15595*, 2024.

Panagiotis Giannoulis, Yorgos Pantis, and Christos Tzamos. Teaching transformers to solve combinatorial problems through efficient trial error, 2026. URL https://arxiv.org/abs/2509.22023.

Jaeyeon Kim, Kulin Shah, and Sitan Chen. Prism: Pre-training discrete diffusion models with auxiliary token-level predictions. *arXiv preprint*, 2025a.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025b.

Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models, 2023. URL https://arxiv.org/abs/2107.00630.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le1, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu3, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

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, 2025. URL https://arxiv.org/abs/2502.09992.

William Peebles and Saining Xie. Scalable diffusion models with transformers. *International Conference on Computer Vision*, 2023.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

David G. Radcliffe. 3 million sudoku puzzles with ratings, 2020. URL https://www.kaggle.com/dsv/1495975. Kaggle dataset.

Yige Tang and Yuchen Zhao. wd1: Reward-weighted diffusion policy optimization for reasoning. *arXiv preprint*, 2025.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. *International Conference on Machine Learning*, 2025a.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. *International Conference on Machine Learning*, 2025b.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025a. URL https://arxiv.org/abs/2506.21734.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Wikipedia contributors. Countdown (game show). https://en.wikipedia.org/wiki/Countdown_(game_show), 2026. Accessed: 2026-02-20.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Wei Bi, and Deyi Xiong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint*, 2024a.

Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Li, Wei Bi, and Deyi Xiong. Diffusion of thoughts: Chain-of-thought reasoning in diffusion language models. *arXiv preprint arXiv:2402.07754*, 2024b.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning, 2025. URL https://arxiv.org/abs/2410.14157.

Yuchen Zhao, Hao Ma, Yijiang Wang, and Jiacheng Ye. D1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint*, 2025.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation, 2024. URL https://arxiv.org/abs/2302.05737.

A Experimental Setup

A.1 Architecture

All models use a Transformer encoder with rotary position embeddings (RoPE). The sequence of length d is embedded via a learned token embedding of dimension D_{hidden} plus a separate clue embedding indicating given positions.

Baseline model. Time conditioning follows the DiT design Peebles & Xie (2023): a sinusoidal timestep embedding is projected to D_{hidden} via a two-layer MLP and injected through Adaptive Layer Normalization (AdaLN) in each Transformer block. Each block applies multi-head self-attention with RoPE, followed by a feed-forward network (expansion ratio 4, GELU activation), with residual connections and dropout. The final layer uses AdaLN followed by a linear projection to vocabulary logits.

Adaptive model. The backbone is identical except that AdaLN time conditioning is replaced by a simple additive time embedding. In addition, the model maintains two parallel hidden streams processed alongside the main token representations:

- **Time prediction head.** An attention-pooled representation of the sequence is passed through a two-layer MLP with sigmoid output, predicting the scalar stopping time $\tau_\theta \in [0, 1]$ (Equation (6)).
- **Mixing prediction head.** A per-position representation, aggregated via masked attention pooling (excluding clue positions), is passed through an MLP producing the per-token confidence $c_\theta^i \in [0, 1]$.

Sudoku hyperparameters. Hidden dimension $D_{\text{hidden}} = 512$, 8 Transformer blocks, 8 attention heads, dropout 0.05, vocabulary size $K = 10$ (digits 0–9), sequence length $d = 81$.

Countdown hyperparameters. Hidden dimension $D_{\text{hidden}} = 512$ and $D_{\text{hidden}} = 256$ for the small network in Table 3, 8 Transformer blocks, 8 attention heads, dropout 0.05. The vocabulary contains $10{,}007$ tokens: integers 0–9,999 (one token each, base-$10{,}000$ encoding), operators $+$, $-$, $\times$, $/$, $=$, and a comma separator (six dedicated tokens), one EOS/padding token, and one mask token. Sequence length $d = 40$. Time conditioning follows the same baseline/adaptive split as Sudoku (AdaLN for the baseline, additive embedding for the adaptive model); the two auxiliary prediction heads are identical in design, with hidden_dim $= 256$ and 3 MLP layers each.

A.2 Training

Baseline training. We train with standard cross-entropy loss on p_1^θ, applied at all positions. The masking schedule is polynomial $\kappa_t = t^2$. We use Adam with learning rate 3×10^{-4}, weight decay 10^{-4}, linear warmup over 1000 steps, gradient clipping at 1.0, and batch size 256. Logits are clipped to $[-50, 50]$.

Adaptive training. The loss combines three terms: (i) standard cross-entropy on the denoiser p_1^θ (applied to all positions, no time weighting), (ii) mixing loss encouraging confident commits only when correct (with $\varepsilon = 0.05$, exponent $k = 1$), and (iii) time prediction MSE. Training uses on-policy states: at each step, we sample x_t from the masking path, take a single no-gradient forward step through the model to obtain $y \sim p_c^{\text{sg}(\theta)}(\cdot \mid x_t)$, and compute the loss on y. Other hyperparameters match the baseline.

A.3 Countdown Training

Baseline training. We minimise the standard cross-entropy loss on p_1^θ, applied at all positions with no time reweighting. The masking schedule is polynomial $\kappa_t = t^2$ (exponent 2). Training uses AdamW with learning rate 5×10^{-4}, weight decay 0.02, linear warmup over 1000 steps, gradient clipping at 1.0, batch size 128, and logit clipping to $[-50, 50]$. Both models are trained for 500k steps. At inference we use 100 Euler steps.

Adaptive training. The loss follows equation 5 with the same three terms as described for Sudoku. The masking schedule is linear $\kappa_t = t$ (exponent 1). On-policy states are generated via a short full-trajectory rollout of 5 steps (applied with probability 0.1): we sample x_t from the masking path, unroll the model for 5 steps to obtain a state z, and compute the loss on z against the ground-truth solution. All other hyperparameters (optimizer, learning rate, batch size, gradient clipping) match the Countdown baseline above.

A.4 Datasets

Sudoku-Extreme. A curated collection of difficult 9×9 Sudoku puzzles from Wang et al. (2025a), each with 17–25 given clues. The dataset contains approximately 50,000 puzzles. Each puzzle is flattened to a sequence of length 81 with vocabulary $\{0, \ldots, 9\}$ where 0 denotes a blank cell. Given clues are enforced throughout training and inference via a binary clue mask.

Kaggle Unfiltered. A large-scale dataset from Radcliffe (2020) of over one million 9×9 Sudoku puzzles with mixed difficulty, sourced from Kaggle. The same preprocessing and representation are used as for Sudoku-Extreme.

Countdown-4. We generate 500k training examples following the setup of Ye et al. (2025); Gandhi et al. (2024): four integers are drawn uniformly, and a target in $[10, 99]$ is produced from a valid arithmetic expression using $+, -, \times, /$ (exact integer division only). Integers from 0 to 9,999 are each encoded as a single token (base-10,000 tokeniser); operators ($+$, $-$, $\times$, $/$, $=$, comma) and the EOS/padding symbol each receive a dedicated token, for a total vocabulary of 10,007 tokens (excluding the mask token, which is appended during training). Each expression is serialised left-to-right and padded to a fixed sequence length of 40. The given numbers (clues) are provided as part of the input and are kept fixed throughout training and inference.

B Comparison Methods

B.1 Top-K and Top-K Margin

Top-K and Top-K Margin Kim et al. (2025b) are inference-time strategies that modify the order in which masked positions are decoded, without retraining the model.

At each denoising step, a certainty score is computed for every masked position i:

- **Top-K:** $\text{certainty}_i = \max_{j \in [K]} p_\theta(x^i = j \mid x_t)$, i.e., the maximum predicted probability.
- **Top-K Margin:** $\text{certainty}_i = p_\theta(x^i = j_1 \mid x_t) - p_\theta(x^i = j_2 \mid x_t)$, where j_1, j_2 are the two most probable tokens. This more reliably captures uncertainty when multiple tokens have similar high probabilities.

The number of positions to unmask is $K_t = M_t \cdot (\alpha_s - \alpha_t)/(1 - \alpha_t)$, where M_t is the current number of masked positions and α_t is the masking schedule. The top-K_t positions by certainty are unmasked, with tokens sampled from the model's predicted distribution. Following Kim et al. (2025b), we add Gumbel noise (scale 0.5) to the certainty scores and use multinomial sampling for token selection.

B.2 ReMDM

Remasking Discrete Diffusion Models (ReMDM) Wang et al. (2025b) modify the backward process of masked diffusion to allow previously decoded tokens to return to the mask state, enabling iterative correction.

The modified transition for unmasked tokens at position i is:

$$q_\sigma(z_s^i \mid z_t^i = x, x_1^i) = (1 - \sigma_t)\, \delta_x(\cdot) + \sigma_t\, \delta_m(\cdot),$$

where σ_t is the remasking probability. A key theoretical property (Theorem 3.1 of Wang et al. (2025b)) is that the marginal $q_\sigma(z_t \mid x_1)$ remains identical to standard masked diffusion, so ReMDM can be applied to any pretrained masked diffusion model without retraining.

We use the rescale schedule $\sigma_t = \eta \cdot \sigma_t^{\max}$ with $\sigma_t^{\max} = \min(1, (1-\alpha_s)/\alpha_t)$ and $\eta = 0.9$, following the recommended defaults.

B.3 GIDD Pretraining

Generalized Interpolating Discrete Diffusion (GIDD) von Rütte et al. (2025b) generalizes the masked diffusion noise process to include uniform noise alongside masking. The conditional forward process is:

$$p_t^{\text{GIDD}}(\cdot \mid x_0, x_1) = \kappa_t^1 \, \delta_{x_0}(\cdot) + \kappa_t^2 \, p_u(\cdot) + \kappa_t^3 \, \delta_{x_1}(\cdot),$$

where p_u is the uniform distribution over the vocabulary, $\sum_i \kappa_t^i = 1$, and the boundary conditions ensure interpolation from x_0 at $t=0$ to x_1 at $t=1$. The uniform component κ_t^2 allows training states to contain random incorrect tokens (rather than only mask or correct tokens), exposing the model to error states during training. However, these are uniformly random errors, not the model's own systematic mistakes.

We train a GIDD baseline using the same backbone architecture, with the hybrid noise schedule recommended in von Rütte et al. (2025b).

C Discrete Flow Matching

For convenience, we reconsider discrete flow matching with our notation based on Gat et al. (2024); Lipman et al. (2024). Just for the notation, for a random variable X_t, $t \in [0,1]$, having a law with density p_{X_t}, we just write p_t, and accordingly

$$p_t(x) = p_{X_t}(x), \quad p_{t|0,1}(x|x_0,x_1) = p_{X_t|(X_0,X_1)=(x_0,x_1)}(x), \tag{8}$$

$$p_{0,1|t}(x_0,x_1|x) = p_{(X_0,X_1)|X_t=x}(x_0,x_1), \quad p_{1|t}(x|x_t) = p_{X_1|X_t=x_t}(x) \tag{9}$$

For a function $u : [0,1] \times \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ which is continuous in time on $(0,1)$ and any initial probability mass function $p_{\text{init}} : \mathcal{S} \to [0,1]$, the **linear ODE** system (Kolmogorov equation)

$$\dot{p}_t(x) = \sum_{y \in \mathcal{S}} u_t(x,y) p_t(y), x \in \mathcal{S} \qquad p_0 = p_{\text{init}} \tag{10}$$

has a unique solution. Supposing the **rate condition**

$$\sum_{x \in \mathcal{S}} u_t(x,y) = 0 \quad \text{for all } y \in \mathcal{S}, \qquad u_t(x,y) \geq 0 \quad \text{for } x \neq y, \tag{11}$$

we obtain that p_t remains a PMF. In this case, we have

$$\sum_{y \in \mathcal{S}} u_t(x,y) p_t(y) = \sum_{y \neq x} \underbrace{u_t(x,y) p_t(y)}_{j_t(x,y)} - \sum_{y \neq x} \underbrace{u_t(y,x) p_t(x)}_{j_t(y,x)} = \sum_{y \in \mathcal{S}} [j_t(x,y) - j_t(y,x)]$$

with flux $j_t(x,y) = u_t(x,y) p_t(y)$ from y to x, so that (p_t, u_t) fulfill the **continuity equation**

$$\dot{p}_t + \text{div}(p_t u_t) = 0, \qquad p_0 = p_{\text{init}} \tag{12}$$

where

$$\text{div}(j_t)(y) = -\sum_{y \in \mathcal{S}} \big(j_t(x,y) - j_t(y,x)\big).$$

In the following, we are exclusively interested in velocity fields fulfilling the rate condition equation 11.

We want to find a flow p_t from the initial distribution to the target distribution, $p_1 = p_{\text{tar}}$. The aim of FM is to learn the velocity field u_t and then to use e.g. the Euler scheme of the ODE equation 10 to sample from p_1.

Let π be a coupling between the source distribution and the target distribution. Then we have by the law of total probability that

$$p_t(x) = \sum_{x_0, x_1 \in \mathcal{S}} p_{t|0,1}(x|x_0,x_1) \pi(x_0,x_1). \tag{13}$$

Using

$$p_{t|0,1}(y|x_0,x_1)\,\pi(x_0,x_1) = p_{0,1|t}(x_0,x_1|y)\,p_t(y),$$

the velocity field to p_t can be deduced as

$$\dot{p}_t(x) = \sum_{x_0,x_1\in\mathcal{S}} \dot{p}_{t|0,1}(x|x_0x_1)\pi(x_0,x_1) \tag{14}$$

$$= \sum_{x_0,x_1\in\mathcal{S}} \sum_{y\in\mathcal{S}} u_t(x,y|x_0,x_1)p_{t|0,1}(y|x_0x_1)\;\pi(x_0,x_1) \tag{15}$$

$$= \sum_{y\in\mathcal{S}} \sum_{x_0,x_1\in\mathcal{S}} u_t(x,y|x_0,x_1)p_{0,1|t}(x_0x_1|y)\,p_t(y). \tag{16}$$

Thus, the velocity field belonging to p_t is

$$u_t(x,y) = \sum_{x_0,x_1\in\mathcal{S}} u_t(x,y|x_0,x_1)p_{0,1|t}(x_0x_1|y). \tag{17}$$

To manage the computational burden, we are only interested in conditional PMF that respect the special structure of our state space $\mathcal{S}$ and restrict ourselves to independent components, i.e., with PMFs $p_t^i : \mathcal{T} \to [0,1]$, we consider

$$p_{t|0,1}(x|x_0,x_1) = \prod_{i=1}^{d} p^i_{t|0,1}(x^i|x_0,x_1)$$

The PMFs $p_t^i(\cdot|x_0,x_1)$ can be defined in several ways. Here we choose

$$p^i_{t|0,1}(x^i|x_0,x_1) = (1-\kappa_t)\delta(x^i,x_0^i) + \kappa_t\delta(x^i,x_1^i). \tag{18}$$

with a monotone increasing, continuously differentiable schedule function $\kappa_t : [0,1] \to [0,1]$ fulfilling $\kappa_0 = 0$ and $\kappa_1 = 1$. The velocity fields corresponding to $p_t^i(\cdot|x_0,x_1)$ can be simply computed by

$$\frac{\mathrm{d}}{\mathrm{d}t}p^i_{t|0,1}(x^i|x_0,x_1) = \dot{\kappa}_t(\delta(x^i,x_1^i) - \delta(x^i,x_0^i)) \tag{19}$$

$$= \dot{\kappa}_t(\delta(x^i,x_1^i) - \frac{p^i_{t|0,1}(x^i|x_0,x_1) + \kappa_t\delta(x^i,x_1^i)}{1-\kappa_t} \tag{20}$$

$$= \frac{\dot{\kappa}_t}{1-\kappa_t}\big(\delta_{x_1^i}(x^i) - p^i_{t|0,1}(x^i|x_0,x_1)\big) \tag{21}$$

$$= \frac{\dot{\kappa}_t}{1-\kappa_t}\sum_{y^i\in\mathcal{T}}\big(\delta(x^i,x_1^i) - \delta(x^i,y^i)\big)p^i_{t|0,1}(y^i|x_0,x_1). \tag{22}$$

Thus, a velocity field belonging to $p^i_{t|0,1}(\cdot|x_0,x_1)$ in equation 10 is given by

$$u_t^i(x^i,y^i|x_0,x_1) = \frac{\dot{\kappa}_t}{1-\kappa_t}\big(\delta(x^i,x_1^i) - \delta(x^i,y^i)\big). \tag{23}$$

Then the vector field $u_t(x,y|x_0,x_1)$ belonging to $p_t(\cdot|x_0,x_1)$ is

$$u_t(x,y|x_0,x_1) = u_t^1(x^1,y^1|x_0,x_1)\,\boldsymbol{\iota_1} \;+\ldots+\; u_t^d(x^d,y^d|x_0,x_1)\,\boldsymbol{\iota_d} \tag{24}$$

$$= \sum_{i=1}^{d} u_t^i(x^i,y^i|x_0,x_1)\,\boldsymbol{\iota_i} \tag{25}$$

where

$$\boldsymbol{\iota_i} = \prod_{\substack{j=1\\ j\neq i}}^{d} \delta(x^j,y^j).$$

Here $\boldsymbol{\iota_i}$ can be seen as a pointer that $u_t^i(x^i,y^i|x_0,x_1)$ appears everywhere in $u_t(x,y|x_0,x_1)$ where $x^j = y^j$ for all $j \neq i$. For a general derivation of vector fields of product measures fulfilling a

continuity equation see Duong et al. (2025).
Reason: for fixed x_0, x_1, let $q_t = p_{t|0,1}(\cdot|x_0, x_1)$ and $v_t = u_t(\cdot|x_0, x_1)$. By the product rule we obtain for $q_t(x) = q_t^1(x^1) \dots q_t^d(x^d)$ that

$$\dot{q}_t(x) = \dot{q}_1(x^1)\frac{q_t(x)}{q_t^1(x^1)} + \ldots + \dot{q}_d(x^d)\frac{q_t(x)}{q_t^d(x^d)} \tag{26}$$

$$= \sum_{y^1 \in \mathcal{T}} v_t^1(x^1, y^1) q_t^1(y^1)\frac{q_t(x)}{q_t^1(x^1)} + \ldots + \sum_{y^d \in \mathcal{T}} v_t^d(x^d, y^d) q_t^d(y^d)\frac{q_t(x)}{q_t^d(x^d)} \tag{27}$$

$$= \sum_{y^1 \in \mathcal{T}} \dots \sum_{y^d \in \mathcal{T}} \Big(v_t^1(x^1, y^1) \prod_{j \neq 1} \delta(x^j, y^j) \; + \ldots + \; v_t^d(x^d, y^d) \prod_{j \neq d} \delta(x^j, y^j) \Big) q_t(y) \tag{28}$$

$$= \sum_{y \in \mathcal{S}} v_t(x, y) q_t(y). \tag{29}$$

By equation 17, equation 24 and equation 23, we obtain

$$u_t(x, y) = \sum_{x_0, x_1 \in \mathcal{S}} \Big(\sum_{i=1}^{d} u_t^i(x^i, y^i | x_0, x_1)\, \boldsymbol{\iota_i} \Big) p_{0,1|t}(x_0, x_1 | y)$$

$$= \frac{\dot{\kappa}_t}{1 - \kappa_t} \sum_{i=1}^{d} \sum_{x_0, x_1 \in \mathcal{S}} \big(\delta(x^i, x_1^i) - \delta(x^i, y^i) \big) p_{0,1|t}(x_0, x_1 | y)\, \boldsymbol{\iota_i}.$$

Since

$$\sum_{x_0, x_1 \in \mathcal{S}} \big(\delta(x^i, x_1^i) - \delta(x^i, y^i) \big) p_{0,1|t}(x_0, x_1 | y) = \sum_{x_1 \in \mathcal{S}} \delta(x^i, x_1^i) p_{1|t}(x_1 | y) - \delta(x^i, y^i) \tag{30}$$

$$= \sum_{x_1^i \in \mathcal{T}} p_{1|t}^i(x_1^i | y) \delta(x^i, x_1^i) - \delta(x^i, y^i), \tag{31}$$

it follows finally that

$$u_t(x, y) = \sum_{i=1}^{d} \frac{\dot{\kappa}_t}{1 - \kappa_t} \big(\sum_{x_1^i \in \mathcal{T}} p_{1|t}^i(x_1^i | y) \delta(x^i, x_1^i) - \delta(x^i, y^i) \big)\, \boldsymbol{\iota_i} \tag{32}$$

$$= \sum_{i=1}^{d} \underbrace{\frac{\dot{\kappa}_t}{1 - \kappa_t} \Big(p_{1|t}^i(x^i | y) - \delta(x^i, y^i) \Big)}_{u_t^i(x^i, y)}\, \boldsymbol{\iota_i}. \tag{33}$$

Therefore we have u_t if we know $p_{1|t}^i$. The Kullback-Leibler divergence of two PMFs $p, q : \mathcal{T} \to [0, 1]$ is defined by

$$\mathrm{KL}(p, q) = \sum_{x \in \mathcal{T}} \log p(x) p(x) - \log q(x) p(x)$$

and their **cross entropy** by

$$\mathrm{CE}(p, q) = -\sum_{x \in \mathcal{T}} \log q(x) p(x) = \mathrm{KL}(p, q) - \sum_{x \in \mathcal{T}} \log p(x) p(x), \tag{34}$$

where we suppose that $p(x) = 0$ whenever $q(x) = 0$, $x \in \mathcal{T}$.

To learn for $i = 1, \dots, d$ the PMF $p_{1|t}^i(\cdot|y)\delta(\cdot, x_1^i)$, we minimize the expectation value of the cross entropy

$$\mathrm{CE}\left(p_{1|t}^i(\cdot|x_t), p_{1|t}^{\theta,i}(\cdot|x_t) \right) = -\sum_{z \in \mathcal{T}} p_{1|t}^i(z|x_t) \log p_{1|t}^{i,\theta}(z|x_t) \tag{35}$$

$$= -\sum_{z \in \mathcal{T}} \sum_{x_0, x_1} \underbrace{p_{1|t,0,1}^i(z|x_t, x_0, x_1)}_{\delta(z, x_1^i)} \pi(x_0, x_1) \log p_{1|t}^{i,\theta}(z|x_t) \tag{36}$$

$$= -\sum_{x_0, x_1} \log p_{1|t}^{i,\theta}(x_1^i|x_t) \tag{37}$$

i.e., we maximize for every $i = 1, \dots, d$ the loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),(x_0,x_1)\sim\pi,x_t\sim p_{t|0,1}}\Big[\sum_{i=1}^{d}\log\left(p_{1|t}^{\theta,i}(x_1^i|x_t)\right)\Big]. \tag{38}$$

For inference, the **Euler forward scheme of the ODE**

$$p_{t+h}(x) = p_t(x) + h\sum_{y\in\mathcal{S}} u_t^\theta(x,y)p_t(y) \tag{39}$$

is computed with the learned velocity $u_t^\theta(x,y)$.