ASYNCHRONOUS FLOW MAPS FOR FLOW LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

The choice of how data elements are ordered and scheduled for modeling has a profound impact on deep generative models. This effect is pronounced in categorical data such as language and code with procedural underlying generative processes. As such, discrete methods, such as autoregressive models and masked diffusion models, can naturally improve by exploiting these orderings during training and inference. However, it is less clear whether this effect can be incorporated into continuous flows for language and categorical data, which have a distinguished advantage of efficient inference through flow map distillation. We introduce asynchronous flows and flow maps, a framework for leveraging token orderings in flow language models by moving each token at a different speed with either prescribed or learned schedules, along with their mathematical formulation and distillation objectives. The resulting asynchronous flow maps are capable of one-step generation while harnessing the benefit of ordered generation by internalizing the procedure into a single forward pass. We train asynchronous flows and flow maps on Sudoku solving, and mathematical coding tasks and achieve state-of-the-art one-step performance.

1 INTRODUCTION

How to order and schedule the generation of data elements in deep generative models is widely studied in different data domains (Chang et al., 2022; Wewer et al., 2025; Baade et al., 2026). These choices determine what information is available when predicting each element and which intermediate prediction problems the model must solve. For categorical data, autoregressive models (ARMs) encode these choices through a prescribed left-to-right factorization, whereas masked diffusion models (MDMs) learn conditional predictions across diverse masking patterns and support different generation orders (Sahoo et al., 2024; Shi et al., 2024). Beyond the order itself, the schedule determines which tokens are updated together and when. In theory, this choice would not matter for ideal models, as every factorization recovers the same joint distribution (Finzi et al., 2026). In practice, however, some prediction problems are harder to learn than others, making generation quality sensitive to the chosen order and schedule (Fathi et al., 2025; Ben-Hamu et al., 2025).

This sensitivity also motivates their use in continuous flow-based language models (Lee et al., 2026; Hu et al., 2026; Chen et al., 2026), which have recently emerged as a promising alternative to discrete diffusion models. A key benefit of these models is that their deterministic generation can be distilled into *flow maps*, replacing iterative generation steps with one or a few model evaluations (Roos et al., 2026; Lee et al., 2026; Potaptchik et al., 2026). However, how to amortize the effect of ordering and scheduling into flow map distillation is underexplored. While existing works bring such ordering and scheduling into continuous generation, distilling those trajectories into flow maps is non-trivial owing to either stochastic discrete jumps included in sampling (Pynadath et al., 2025; Jutras-Dubé et al., 2026; Azangulov et al., 2026) or trajectory restarts that redraw the noise (Agarwal et al., 2026).

In this paper, we introduce **Asynchronous Flow Maps**, a framework for learning one-step generators from asynchronous generation of flow language models. Building on prior work across domains (Fathi et al., 2025; Qian & Cheng, 2026; Bansal & Sanghavi, 2026; Rekesh et al., 2026), we formulate the generation dynamics in which different tokens denoise at different rates governed by a scheduler. This scheduler unifies synchronous generation (Lee et al., 2026), schedules prescribed in advance (Bansal & Sanghavi, 2026), and learned schedules that adapt to intermediate predictions

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(Chang et al., 2022; Ben-Hamu et al., 2025). Since the rates may depend on the tokens themselves, we treat the denoising times of each token as part of the dynamic state and describe the evolution of tokens and their denoising times as a single coupled dynamics. We then characterize the flow map of these coupled dynamics and develop a distillation objective that jointly captures token evolution and scheduling between arbitrary timesteps. By distilling state-dependent schedules into asynchronous flow maps, we achieve state-of-the-art one-step generation performance among existing diffusion and flow language models on Sudoku solving and TinyGSM. Our main contributions are:

- **Unified formulation of asynchronous generation.** We formulate asynchronous generation of flow language models as coupled dynamics of tokens and their tokenwise times governed by a scheduler, which unifies the synchronous schedule, prescribed schedules, and state-dependent schedules ported from MDM samplers as special cases (Sections 3.1 and 3.2).
- **Asynchronous flow maps.** We characterize the flow map of these coupled dynamics and derive a distillation objective that learns it as a one-step generator, both when the schedule is unknown at training time and adapts during generation, and when it is prescribed in advance, in which case the objective simplifies to training from scratch (Sections 3.3 and 3.4).
- **Empirical gains in one-step generation.** Asynchronous flow maps achieve state-of-the-art one-step performance on Sudoku and GSM8K, by internalizing the effect of ordering and scheduling into a single forward pass (Section 5).

2 Background

Flow language models. Continuous flow-based language models (Lee et al., 2026; Hu et al., 2026; Chen et al., 2026) model discrete token sequences by learning continuous flows from a noise distribution to continuous representations of data. Let V be a vocabulary of tokens and $\mathbf{y} = (\mathbf{y}_i)_{i=1}^L \in V^L$ a token sequence. Following Lee et al. (2026), we represent each token by its one-hot encoding, yielding $\mathbf{x}_1 = (\mathsf{onehot}(\mathbf{y}_i))_{i=1}^L \in \mathbb{R}^{L\times|V|}$. Let p_1 denote the distribution of these one-hot representations and $p_0 = \mathsf{N}(0, I)$ the Gaussian noise distribution on the same space. We define a probability path by interpolating between independent draws $\mathbf{x}_0 \sim p_0$ and $\mathbf{x}_1 \sim p_1$:

$$I_t \coloneqq (1-t)\mathbf{x}_0 + t\mathbf{x}_1, \qquad p_t \coloneqq \mathsf{Law}(I_t), \qquad t \in [0,1]. \tag{1}$$

The probability path admits a deterministic representation,

$$\dot{\mathbf{x}}_t = b_t(\mathbf{x}_t), \qquad b_t(\mathbf{x}) = \mathbb{E}[\mathbf{x}_1 - \mathbf{x}_0 | I_t = \mathbf{x}], \tag{2}$$

where b_t is the velocity field of the probability flow. Instead of predicting the velocity itself, it is common to learn the denoiser, which predicts the posterior mean of the clean data:

$$D_t(\mathbf{x}) \coloneqq \mathbb{E}[\mathbf{x}_1 | I_t = \mathbf{x}], \qquad b_t(\mathbf{x}) = \frac{D_t(\mathbf{x}) - \mathbf{x}}{1-t}. \tag{3}$$

Since $\mathbf{x}_1$ is one-hot, $D_t^i(\mathbf{x})$ is a probability distribution over V at each position i, and the denoiser can be learned by minimizing a tokenwise cross-entropy loss $\mathcal{L}_{\mathsf{CE}}(\hat{D}) \coloneqq \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1} \sum_i [-\log \hat{D}_t^i(I_t)]_{\mathbf{y}_i}$ (Dieleman et al., 2022; Eijkelboom et al., 2024; Lee et al., 2026).

Flow map language models. The deterministic flow given in (2) defines a unique flow map $X_{s,t} : \mathbb{R}^{L\times|V|} \to \mathbb{R}^{L\times|V|}$, the solution operator of the flow satisfying

$$X_{s,t}(\mathbf{x}) = \mathbf{x} + \int_s^t b_r(X_{s,r}(\mathbf{x}))\mathrm{d}r = \mathbf{x} + (t-s)v_{s,t}(\mathbf{x}), \tag{4}$$

where $v_{s,t}$ is the average velocity between s and t, satisfying $v_{s,s} = b_s$. By learning the flow map, we can jump between arbitrary times, and evaluating it with $s = 0$ and $t = 1$ gives a one-step jump from noise to data. Recent work on learning flow maps over categorical data (Roos et al., 2026; Lee et al., 2026; Potaptchik et al., 2026) parameterizes it via the two-time denoiser $\delta_{s,t} : \mathbb{R}^{L\times|V|} \to (\Delta^{|V|-1})^L$,

$$\delta_{s,t}(\mathbf{x}) \coloneqq \mathbf{x} + (1-s)v_{s,t}(\mathbf{x}), \tag{5}$$

which extrapolates the flow map displacement to $t = 1$ and satisfies $\delta_{s,s}(\mathbf{x}) = D_s(\mathbf{x})$. Since it is simplex-valued, $\delta_{s,t}$ can be learned by self-distillation with KL divergence,

$$\mathcal{L}_{\mathsf{KL}}(\hat{\delta}) = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}(D_t(I_t) \,\|\, \hat{\delta}_{t,t}(I_t))\right] + \mathbb{E}_{s,u,t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}(\bar{\delta}_{s,t} \,\|\, \hat{\delta}_{s,t}(I_s))\right], \tag{6}$$

where, for $s < u < t$ and with the stop-gradient operator $\mathsf{sg}(\cdot)$,

$$\bar{\delta}_{s,t} = \mathsf{sg}(\gamma\, \hat{\delta}_{s,u}(I_s) + (1-\gamma)\, \hat{\delta}_{u,t}(\hat{X}_{s,u}(I_s))), \qquad \gamma = \tfrac{(1-t)(u-s)}{(1-u)(t-s)}, \tag{7}$$

where $\hat{X}_{s,t}(\mathbf{x}) = \frac{1-t}{1-s}\mathbf{x} + \frac{t-s}{1-s}\hat{\delta}_{s,t}(\mathbf{x})$ is the flow map induced by the learned denoiser $\hat{\delta}$.

3 Asynchronous Flow Maps

In this section, we formalize asynchronous generation of categorical data by assigning each token its own time, which advances at a rate defined by a scheduler. Since an effective schedule is often not known at training time in practice, we develop a general formulation that allows us to train a denoiser over all tokenwise times and then use it to define the generation dynamics under any chosen schedule. Finally, we show how to distill these dynamics into a flow map for few-step generation.

3.1 Training the denoiser with tokenwise times

We define asynchronous generation as denoising different token positions at different rates. To represent their progress, we introduce a vector of times

$$\tau = (\tau^1, \ldots, \tau^L) \in [0,1]^L, \tag{8}$$

where τ^i is the time of token position i. We call the evolution of tokenwise times during generation a *schedule*, and specify it through a *scheduler* in Section 3.2. Similar to the interpolant of (1), we associate $\tau = \mathbf{0}$ with noise and $\tau = \mathbf{1}$ with clean data, and interpolate between them tokenwise. For noise $\mathbf{x_0} \sim p_{\mathbf{0}} = \mathsf{N}(0, I)$ and data $\mathbf{x_1} \sim p_{\mathbf{1}} = p_1$, we define the asynchronous interpolant

$$I_\tau := (\mathbf{1} - \tau)\mathbf{x_0} + \tau \mathbf{x_1}, \qquad p_\tau := \mathsf{Law}(I_\tau). \tag{9}$$

Hereafter, we abuse notation and write $(\tau\, \mathbf{x})^i = (\tau \odot \mathbf{x})^i := \tau^i \mathbf{x}^i$ and $(\frac{\mathbf{x}}{\tau})^i := \frac{\mathbf{x}^i}{\tau^i}$ for $\tau \in \mathbb{R}^L$ and $\mathbf{x} \in \mathbb{R}^{L \times |V|}$, and apply the same convention to elementwise operations between vectors in $\mathbb{R}^L$.

For each τ, we define the corresponding denoiser as the conditional mean of the clean data:

$$D_\tau(\mathbf{x}) := \mathbb{E}[\mathbf{x_1} \mid I_\tau = \mathbf{x}]. \tag{10}$$

In general, an effective schedule and tokenwise times it generates are not known at training time. We therefore train over all $\tau \in [0,1]^L$, parameterizing the family of denoisers with a single network $\hat{D}_\tau(\mathbf{x})$. In practice, this can be done with diffusion forcing (Chen et al., 2024), which samples each τ^i independently from $\mathsf{U}[0,1]$ during training. Since $\hat{D}$ predicts a probability distribution over the vocabulary at each position, we can train it by minimizing tokenwise cross-entropy:

Proposition 1. *Consider the objective*

$$\mathcal{L}_{\mathsf{CE}}(\hat{D}) := \mathbb{E}_{\tau, \mathbf{x_0}, \mathbf{x_1}} \sum_{i=1}^{L} \Big[-\log \hat{D}^i_\tau(I_\tau) \Big]_{\mathbf{y}_i}, \qquad \tau^i \overset{\text{iid}}{\sim} \mathsf{U}[0,1]. \tag{11}$$

Any population minimizer $\hat{D}^\star$ satisfies $\hat{D}^\star_\tau = D_\tau$ almost surely under the training distribution.

A proof is given in Appendix A.1. Thus, training with diffusion forcing recovers the denoiser D_τ.

3.2 Asynchronous generation with schedulers

Having established the denoiser D_τ on asynchronously noised data, we now consider how to generate samples. We first note that this denoiser induces a *notion* of velocity $b_\tau(\mathbf{x})$ analogous to (2) as the expected direction towards clean data evaluated at the given asynchronously noised data $\mathbf{x}$:

$$b_\tau(\mathbf{x}) := \mathbb{E}[\mathbf{x_1} - \mathbf{x_0} \mid I_\tau = \mathbf{x}] = \frac{D_\tau(\mathbf{x}) - \mathbf{x}}{\mathbf{1} - \tau}, \tag{12}$$

Intuitively, we can generate a sample $\mathbf{x_1} \sim p_{\mathbf{1}}$ by starting from noise $\mathbf{x_0} \sim p_{\mathbf{0}}$ and then progressively moving $\mathbf{x}_\tau$ along the direction suggested by $b_\tau(\mathbf{x}_\tau)$ (12) until we reach the terminal time $\tau = \mathbf{1}$.

This idea is however problematic. Unlike in standard flows (2) where we only need to progress a scalar time t from 0 to 1, and there is only one possible way to move (up to reparameterization), there are infinitely many possible choices of how tokenwise time τ can progress from $\mathbf{0}$ to $\mathbf{1}$:

Proposition 2. *There is exactly one continuous nondecreasing path of a scalar time t from 0 to 1 up to a continuous nondecreasing reparameterization. For tokenwise times $\tau \in [0,1]^L$ with $L \geq 2$, there are infinitely many such paths from $\mathbf{0}$ to $\mathbf{1}$.*

A proof is in Appendix A.2. Therefore, to make sense of a generative trajectory $\mathbf{x}_\tau$ from $\mathbf{x_0}$ to $\mathbf{x_1}$, we must make a nonambiguous choice of how τ moves from $\mathbf{0}$ to $\mathbf{1}$. Here, given the nonambiguity of scalar time, we specify the movement of τ itself as a one-parameter curve $\tau(t)$ across $t \in [0,1]$ under boundary conditions $\tau(0) = \mathbf{0}, \tau(1) = \mathbf{1}$. With this choice, a generative trajectory moves $\mathbf{x}_{\tau(t)}$ from $t = 0$ to 1, using t as *base time*, effectively making asynchronous generation nonambiguous.

There are many ways to specify $\tau(t)$. Since there are only two objects $\mathbf{x}_{\tau(t)}$ and $\tau(t)$ available, a natural choice is to use a dynamics on these two states, by choosing a function ω we call a scheduler:

$$\dot{\tau}(t) = \omega_t(\mathbf{x}_{\tau(t)}, \tau(t)), \qquad t \in [0,1]. \tag{13}$$

On a schedule, token i of the interpolant (9) depends only on τ^i, so it moves as $\frac{d}{dt} I_{\tau_t} = \dot{\tau}_t(\mathbf{x_1} - \mathbf{x_0})$. Replacing the unknown $\mathbf{x_1} - \mathbf{x_0}$ by its conditional mean $b_{\tau_t}(\mathbf{x}_t)$ from (12) and $\dot{\tau}_t$ by (13), gives

$$\dot{\mathbf{x}}_t = b_t(\mathbf{x}_t, \tau_t), \qquad b_t(\mathbf{x}, \tau_t) \coloneqq \dot{\tau}_t b_{\tau_t}(\mathbf{x}) = \omega_t(\mathbf{x}, \tau_t) b_{\tau_t}(\mathbf{x}), \tag{14}$$

which scales each position of b_τ by the scheduler ω. We shortly discuss the consequences of this approximation. Notably, this makes (13) and (14) a dynamics on a coupled state:

$$\mathbf{z}_t \coloneqq (\mathbf{x}_t, \tau_t), \qquad \dot{\mathbf{z}}_t = \begin{bmatrix} b_t(\mathbf{x}_t, \tau_t) \\ \omega_t(\mathbf{x}_t, \tau_t) \end{bmatrix}, \qquad \mathbf{z}_0 = (\mathbf{x}_0, \mathbf{0}), \qquad \mathbf{x}_0 \sim p_0. \tag{15}$$

Every asynchronous generation procedure we consider is thus a single deterministic ODE on $\mathbf{z}_t$, differing only in the choice of ω, which later allows us to define its flow map (Section 3.3).

Prescribed schedules. The underlying approximation of (15) distinguishes two types of schedulers, depending on whether it depends on the current generation $\mathbf{x}$. When the scheduler does not depend on $\mathbf{x}$, then tokenwise times become a pure function of the base time, $\tau_t = \tau(t)$, for a path $\tau : [0,1] \to [0,1]^L$ shared for all samples. Since such a path needs to be determined perfectly before generation, we call such $\tau(t)$ a *prescribed schedule*. These schedules are useful when a good generative procedure is known, for instance left-to-right ordering for natural language (Wu et al., 2023). Then (14) and (15) are no longer an approximation and recovers the true probability flow:

Proposition 3. *Let $\tau(t)$ be a continuous prescribed schedule with $\tau(0) = \mathbf{0}$ and $\tau(1) = \mathbf{1}$. Then the coupled dynamics in* (15) *reduce to*

$$\dot{\mathbf{x}}_t = b_t(\mathbf{x}_t, \tau(t)), \qquad \mathbf{x}_0 \sim p_0, \tag{16}$$

and satisfy $\mathbf{x}_t \sim p_{\tau(t)}$ for every $t \in [0,1)$, with $p_{\tau(t)} \to p_1$ as $t \uparrow 1$.

A proof is in Appendix A.3. Therefore, with an exact denoiser, *any* prescribed asynchronous schedule recovers the data distribution p_1. Since this only requires the denoiser along $\tau(t)$, we may train it only on these tokenwise times instead of arbitrary samples as in (11), as we exploit in Section 3.4.

Learned schedules. If the scheduler ω depends on the current generation $\mathbf{x}$, tokenwise times are determined *during* generation. These are useful when prescribed schedules are not available, and so we would like to use an estimation such as confidence of the learned denoiser to schedule asynchronous generation. The tradeoff is that Proposition 3 no longer applies, so there is no guarantee that the true denoiser reproduces the data distribution. Nevertheless, it is useful in practice where we never learn the true denoiser. As such, we call these schedules *state-dependent*, or *learned* reflecting their use-cases.

Even in this case, the training distribution of (I_τ, τ) has full support since $\mathbf{x}_0$ is Gaussian and each $\tau^i \sim \mathsf{U}[0,1]$. Therefore, every state visited by (15) lies within the training range for any choice of ω, so a single denoiser trained with diffusion forcing can be used with any scheduler at inference time.

In practice, choosing a good scheduler is important. Since a trained denoiser only approximates D, with prediction errors varying across positions and states, a good scheduler can improve generation

by presenting the model with easy denoising problems at each timestep (Kim et al., 2025). A natural choice is to advance the positions that the denoiser predicts confidently. This simple idea allows us to adapt existing unmasking criteria of masked diffusion models, as they are centered around designing a good heuristic of measuring denoiser confidence. We consider two specific choices (see Appendix C for more examples and details):

- **Top-k confidence scheduler** advances the tokenwise times of the k most confident positions faster, and gradually increases k from 0 to L, following MaskGIT (Chang et al., 2022).
- **Entropy-bounded scheduler** advances the tokenwise times of the most confident positions faster until their total predicted entropy exceeds a threshold, keeping the total uncertainty bounded, following EB-Sampler (Ben-Hamu et al., 2025).

3.3 Asynchronous flow maps

In Section 3.2, we have formalized the dynamics of asynchronous generation obtained from the denoiser and a scheduler. We now define the flow map of these dynamics and a self-distillation objective for learning it, so that asynchronous generation can be carried out in a few steps.

Proposition 4. *On any interval where* (15) *admits unique solutions, the* ***asynchronous flow map****, i.e., its solution operator $F_{s,t}(\mathbf{z}_s) = \mathbf{z}_t$, satisfies*

$$F_{s,t}(\mathbf{z}_s) = \mathbf{z}_s + \int_s^t \dot{\mathbf{z}}_r \,\mathrm{d}r, \qquad \textit{and} \qquad F_{s,t} = F_{u,t} \circ F_{s,u}, \qquad s \le u \le t < 1. \tag{17}$$

A proof is given in Appendix A.4. Thus $F_{s,t}$ has the standard flow map properties, and its semigroup condition (17) gives a self-distillation target for learning it, as in (6).

To parameterize $F_{s,t}$, we must predict both the token states and their tokenwise times at t. For $s < t$, define the average tokenwise time rate and average velocity along the trajectory by

$$\omega_{s,t}(\mathbf{z}_s) = \frac{1}{t-s}\int_s^t \omega_r(\mathbf{x}_r, \tau_r)\,\mathrm{d}r, \qquad v_{s,t}(\mathbf{z}_s) = \frac{1}{t-s}\int_s^t b_r(\mathbf{x}_r, \tau_r)\,\mathrm{d}r. \tag{18}$$

The integral of b_r can be parameterized using the two-time denoiser in (5), extended to tokenwise times, while the integral of ω determines the tokenwise times reached at t:

$$\tau_{s,t}(\mathbf{z}_s) \coloneqq \tau_s + (t-s)\omega_{s,t}(\mathbf{z}_s), \qquad \delta_{s,t}(\mathbf{z}_s) \coloneqq \mathbf{x}_s + \frac{\mathbf{1}-\tau_s}{\omega_{s,t}(\mathbf{z}_s)} v_{s,t}(\mathbf{z}_s). \tag{19}$$

This parameterization satisfies the following:

Proposition 5. *Let $F_{s,t}$ be the flow map of* (15)*. The parameterizations of* (19) *satisfy:*

(i) The flow map is recovered as $F_{s,t}(\mathbf{z}) = (F^x_{s,t}(\mathbf{z}), \tau_{s,t}(\mathbf{z}))$ with

$$F^x_{s,t}(\mathbf{z}) = \mathbf{x} + \frac{\tau_{s,t}(\mathbf{z}) - \tau}{\mathbf{1}-\tau}(\delta_{s,t}(\mathbf{z}) - \mathbf{x}). \tag{20}$$

(ii) On the diagonal $s = t$,

$$\delta_{s,s}(\mathbf{z}) = D_\tau(\mathbf{x}), \qquad \tau_{s,s}(\mathbf{z}) = \tau, \qquad \omega_{s,s}(\mathbf{z}) = \omega_s(\mathbf{x}, \tau). \tag{21}$$

(iii) For $s < u < t$, let $\mathbf{z}_u = F_{s,u}(\mathbf{z})$. The semigroup condition (17) *is equivalent to*

$$\tau_{s,t}(\mathbf{z}) = \tau_{u,t}(\mathbf{z}_u), \qquad \delta_{s,t}(\mathbf{z}) = \gamma\delta_{s,u}(\mathbf{z}) + (\mathbf{1}-\gamma)\delta_{u,t}(\mathbf{z}_u), \tag{22}$$

where

$$\gamma = \frac{(\mathbf{1}-\tau_{u,t}(\mathbf{z}_u)) \odot (\tau_{s,u}(\mathbf{z}) - \tau)}{(\mathbf{1}-\tau_{s,u}(\mathbf{z})) \odot (\tau_{u,t}(\mathbf{z}_u) - \tau)} \in [0,1]^L. \tag{23}$$

Moreover, $\delta^i_{s,t}(\mathbf{z})$ is simplex-valued whenever D^i is simplex-valued.

A proof is in Appendix A.5. Let $\hat{\delta}_{s,t}$ and $\hat{\omega}_{s,t}$ denote the student's predictions of $\delta_{s,t}$ and $\omega_{s,t}$, with destination tokenwise times $\hat{\tau}_{s,t} = \tau_s + (t-s)\,\hat{\omega}_{s,t}$. We enforce (ii) and (iii) of Proposition 5 using

$$\begin{aligned}\mathcal{L}(\hat{\delta},\hat{\omega}) = \mathbb{E}_{t,\mathbf{z}_t}\left[\mathsf{KL}(D_{\tau_t}(\mathbf{x}_t)\,\|\,\hat{\delta}_{t,t}(\mathbf{z}_t)) + \lambda\cdot\|\hat{\omega}_{t,t}(\mathbf{z}_t) - \omega_t(\mathbf{x}_t,\tau_t)\|^2\right] \\ + \mathbb{E}_{s,u,t,\mathbf{z}_s}\left[\mathsf{KL}(\bar{\delta}_{s,t}\,\|\,\hat{\delta}_{s,t}(\mathbf{z}_s)) + \lambda\cdot\|\bar{\tau}_{s,t} - \hat{\tau}_{s,t}(\mathbf{z}_s)\|^2\right].\end{aligned} \tag{24}$$

The first term matches the instantaneous dynamics of both the token states and tokenwise times (ii). The second term enforces the composition identities (iii) on both components through the targets

$$\begin{aligned}\bar{\tau}_{s,t} &= \mathsf{sg}\left[\hat{\tau}_{u,t}(\hat{F}_{s,u}(\mathbf{z}_s))\right] \\ \bar{\delta}_{s,t} &= \mathsf{sg}\left[\gamma\hat{\delta}_{s,u}(\mathbf{z}_s) + (\mathbf{1}-\gamma)\hat{\delta}_{u,t}(\hat{F}_{s,u}(\mathbf{z}_s))\right], \qquad \gamma = \tfrac{(\mathbf{1}-\bar{\tau}_{s,t})\odot(\hat{\tau}_{s,u}-\tau_s)}{(\mathbf{1}-\hat{\tau}_{s,u})\odot(\bar{\tau}_{s,t}-\tau_s)},\end{aligned} \tag{25}$$

where $s < u < t$ and $\hat{F}_{s,u}$ is the flow map induced by the student through (20). At the population optimum, this objective recovers the flow map $F_{s,t}$ (Appendix B.1).

Inference. We generate samples by replacing numerical integration of (15) with learned transitions, on a time grid $0 = t_0 < \cdots < t_N = 1$

$$\hat{\mathbf{z}}_{t_{n+1}} = \hat{F}_{t_n,t_{n+1}}(\hat{\mathbf{z}}_{t_n}), \qquad n = 0,\ldots,N-1, \qquad \hat{\mathbf{z}}_0 = (\mathbf{x}_0,\mathbf{0}), \tag{26}$$

and decode via tokenwise argmax on the two-time denoiser output $\hat{\delta}_{t_{N-1},1}(\hat{\mathbf{z}}_{t_{N-1}})$. For $N = 1$, a single evaluation of $\hat{\delta}_{0,1}$ generates the entire sequence, without evaluating the scheduler.

3.4 Implementation

We now describe the practical implementation of asynchronous flow maps.

Scheduler implementation. For implementing the scheduler, we use the form

$$\omega^i_t(\mathbf{x},\tau) = \frac{1-\tau^i}{1-t}\,k^i_t(\mathbf{x},\tau), \qquad t < 1, \tag{27}$$

where a positive rate multiplier $k^i > 0$ ensures that tokenwise times increase strictly, and a lower bound $k^i \geq k_{\min} > 0$ near $t = 1$ ensures that they reach one. Setting $k^i = 1$ recovers the synchronous schedule. Prescribed schedules correspond to k^i that depends only on t, while the other schedulers raise k^i for the positions they select (see Appendix C for details).

Time conditioning. We extend the scalar time conditioning of FLMs to the vector of tokenwise times τ by applying a shared time-embedding layer to each tokenwise time τ^i, so that every position receives its own conditioning vector. For the flow map, we additionally embed the base times s and t with zero-initialized layers following Lee et al. (2026), and add them to the tokenwise time embeddings of τ_s, which modulate each block through adaLN (Peebles & Xie, 2023).

Parameterization of tokenwise time. Besides the two-time denoiser $\hat{\delta}_{s,t}$, the flow map must predict the tokenwise times $\hat{\tau}_{s,t}$ reached at t (Section 3.3), which must satisfy $\tau_s \leq \hat{\tau}_{s,t} < \mathbf{1}$ for $t < 1$ and $\hat{\tau}_{s,1} = \mathbf{1}$. To guarantee this, the model instead predicts an average rate multiplier $\hat{\kappa}_{s,t} > 0$, the two-time counterpart of k_t in (27), and sets

$$\hat{\tau}_{s,t}(\mathbf{z}_s) = \mathbf{1} - (\mathbf{1}-\tau_s)\odot\left(\frac{1-t}{1-s}\right)^{\hat{\kappa}_{s,t}(\mathbf{z}_s)}, \tag{28}$$

which integrates (27) exactly over $[s,t]$ when $\kappa_{s,t}$ is the weighted average of k along the trajectory (see Appendix B.2 for details). We implement the tokenwise time diagonal term of (24) as $\lambda\|k_t(\mathbf{x}_t,\tau_t) - \hat{\kappa}_{t,t}(\mathbf{z}_t)\|^2$, which is equivalent to matching $\hat{\omega}_{t,t} = \omega_t(\mathbf{x}_t,\tau_t)$.

Dual-head architecture. To make a single model jointly predict the two-time denoiser $\hat{\delta}_{s,t}$ and the average rate multiplier $\hat{\kappa}_{s,t}$, we attach two output heads to a shared backbone. We initialize

the backbone and the two-time denoiser head from the diffusion-forcing teacher D_τ, and add a new tokenwise time head on the features of the last hidden layer, whose output is exponentiated to ensure $\hat{\kappa}_{s,t} > 0$. The new head is initialized to zero, so that distillation starts from the teacher's predictions with the synchronous rate $\hat{\kappa}_{s,t} = \mathbf{1}$.

Rollout curriculum training. Under a state-dependent ω, the tokenwise times in $\mathbf{z}_s$ depend on the preceding trajectory, so $\mathbf{z}_s$ cannot be sampled from an interpolant. We therefore initially obtain it by rolling out the teacher D_τ under ω from $\mathbf{z}_0 = (\mathbf{x}_0, \mathbf{0})$ to time s. Since these rollouts are expensive, we switch to the current student flow map after an initial training round and halve the number of rollout steps at successive rounds, where the fixed teacher continues to supervise the diagonal terms.

Flow maps for prescribed schedules. When a useful prescribed schedule $\tau(t)$ is known in advance, the token states at any time s follow the known interpolant distribution $p_{\tau(s)}$ (Proposition 3). We can therefore sample the interpolant directly from $I_{\tau(s)}$, eliminating both teacher rollouts and the need to predict tokenwise times. It greatly simplifies the training of an asynchronous flow map, enabling training from scratch with clean data as a diagonal target

$$\mathbb{E}\left[\mathsf{KL}(\mathbf{x}_1 \,\|\, \hat{\delta}^{\tau}_{t,t}(I_{\tau(t)}))\right] + \mathbb{E}\left[\mathsf{KL}(\bar{\delta}^{\tau}_{s,t} \,\|\, \hat{\delta}^{\tau}_{s,t}(I_{\tau(s)}))\right], \tag{29}$$

where $\bar{\delta}^{\tau}_{s,t}$ is the composition target of (25) evaluated along the schedule (Appendix B.3).

4 Related works

Asynchronous flows and diffusion. Asynchronous flows and diffusions denoise coupled components at different speeds. In image generation, Baade et al. (2026) study asynchronous latent–pixel generation under both single- and multi-schedule training, while Qian & Cheng (2026) learn schedules for semantic and texture latents that remain fixed during generation. For text, Wu et al. (2023) propose a left-to-right schedule, and for reasoning over continuous variables, Wewer et al. (2025) order the variables by the uncertainty of the denoiser during generation. We formulate prescribed and state-dependent tokenwise scheduling within a common continuous dynamics and derive the corresponding flow maps and distillation objectives.

Flow language models and tokenwise times. Flow language models (Lee et al., 2026; Chen et al., 2026; Hu et al., 2026) generate discrete tokens with flows on continuous embeddings, and their flow maps enable one- and few-step generation (Lee et al., 2026; Roos et al., 2026; Potaptchik et al., 2026). While most evolve the tokens with a single time shared by the whole sequence, some works instead use tokenwise times. Bansal & Sanghavi (2026) fix token ranks using the initial prediction entropy and distill the resulting dynamics with the destination token times supplied as conditioning. Rekesh et al. (2026) instead adjust tokenwise step sizes using the current prediction confidence at every step, but do not distill the resulting dynamics. We instead jointly model token states and their tokenwise times, enabling the student to predict both and thereby distill generation dynamics with state-dependent schedules.

Ordering in diffusion language models. The importance of generation order is well established in discrete diffusion, where performance often depends critically on the order of tokens (Kim et al., 2025; Nie et al., 2026). Aligning training with inference-time planning improves generation across tasks (Peng et al., 2026), while adaptive selection of tokens to unmask together reduces sampling steps (Ben-Hamu et al., 2025; Kim et al., 2026). For continuous language models, generation order has been introduced through the commitment order of positions (Jutras-Dubé et al., 2026; Azangulov et al., 2026; Agarwal et al., 2026). However, these methods commit tokens outside a continuous deterministic trajectory, through stochastic discrete commitments or trajectory restarts, so their generation process cannot be directly distilled into a flow map. This motivates our framework which incorporates generation order into the trajectories being distilled.

5 Experiments

Throughout the experiments, we address two questions:

Q1. Can asynchronous flows leverage tokenwise schedules for better generation? (Section 5.1)
Q2. Can asynchronous flow maps leverage these schedules for one-step generation? (Section 5.2)

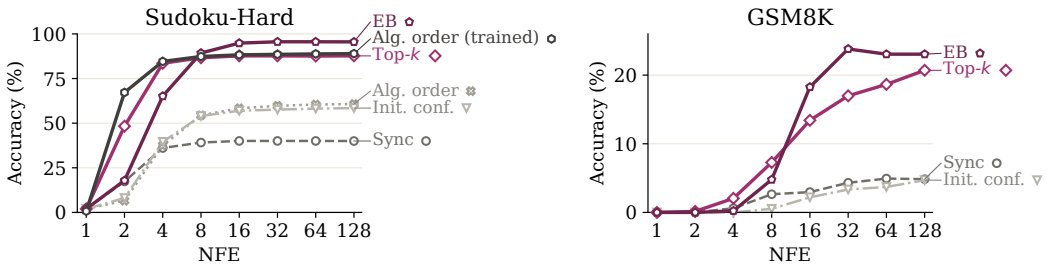

Figure 1: Accuracy under different schedulers on Sudoku-Hard (left) and GSM8K (right). A well-chosen asynchronous schedule improves performance over the synchronous schedule (Sync).

Setup. We evaluate our method on mathematical code generation and Sudoku solving. For mathematical code generation, we train on code completions from TinyGSM (Liu et al., 2023) and measure zero-shot accuracy (%) on the full GSM8K (Cobbe et al., 2021) test set, following Kim et al. (2025). For Sudoku, we use the dataset with three difficulty levels and report exact solving accuracy (%) on the full test set, following Deschenaux & Gulcehre (2026). We denote the asynchronous flow and flow map models by aFLM and aFMLM, respectively.

We compare against one-step baselines and report multi-step results to contextualize one-step performance. The one-step baselines are FMLM (Lee et al., 2026) and FMLM+ (Agarwal et al., 2026). For reference, we also include multi-step results of discrete diffusion models (MDLM (Sahoo et al., 2024) and Duo (Sahoo et al., 2025)) and continuous methods (CANDI (Pynadath et al., 2025), FLM (Lee et al., 2026), and S-FLM (Deschenaux & Gulcehre, 2026)), with the choice of baselines informed by Deschenaux & Gulcehre (2026).

5.1 Effect of scheduling on asynchronous flows (Q1)

For each dataset, we train a single teacher denoiser with diffusion forcing (Chen et al., 2024) and apply different schedulers without retraining. We compare the synchronous schedule (Sync) as in FLM (Lee et al., 2026), a schedule fixed by the initial prediction confidence (Init. conf.), a variant of Bansal & Sanghavi (2026), and state-dependent schedulers including top-k confidence scheduling (Top-k) and entropy-bounded scheduling (EB) (Chang et al., 2022; Ben-Hamu et al., 2025). We evaluate the effect of scheduling while keeping the denoiser fixed.

Figure 1 shows that state-dependent scheduling substantially improves generation accuracy on both tasks. At 32 NFE, EB improves the accuracy over Sync from 40.1% to 95.6% on Sudoku-Hard and from 4.3% to 23.8% on GSM8K. At the same budget, both Top-k and EB outperform Init. conf., suggesting that adapting the schedule to intermediate predictions can be more effective than fixing it from the initial prediction. These results show that scheduling strategies developed for discrete diffusion can also improve generation accuracy in asynchronous flows. Our framework thus allows continuous flow models to benefit from these strategies without retraining the denoiser.

aFLM with prescribed schedule. Given a known generation schedule, we test whether training the denoiser only along it outperforms diffusion forcing when both generate with this schedule. On Sudoku, we construct an algorithmic-order schedule (Alg. order) that denoises cells with low uncertainty faster. Details are provided in Appendix C. We train a denoiser only along this schedule without diffusion forcing (Proposition 3), denoted Alg. order (trained). This model substantially outperforms the diffusion-forcing denoiser under the same schedule (Figure 1), demonstrating the benefit of concentrating training on the schedule used for generation. Nevertheless, the diffusion-forcing denoiser with EB achieves higher accuracy at larger NFEs, showing that effective scheduling can also be achieved from the model's predictions without a task-specific order.

5.2 One-step generation with asynchronous flow maps (Q2)

We distill aFLM from Section 5.1 under EB and synchronous scheduling to obtain aFMLM (EB) and aFMLM (Sync), respectively. Using the same teacher for both variants on each task allows us to assess the contribution of asynchronous scheduling during distillation.

Effects of asynchronous scheduling and distillation. We assess the contributions of scheduling and distillation through controlled comparisons and contextualize the resulting one-step performance against existing baselines. At 1 NFE, aFMLM (EB) achieves 15.9% accuracy on Sudoku-

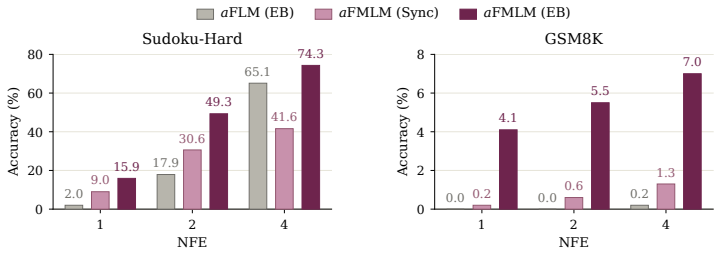

Figure 2: Accuracy (%) on Sudoku-Hard (left) and GSM8K (right). Both aFLM and aFMLM adopts entropy-bounded (EB) scheduling for sampling and distillation, respectively.

Table 1: Exact solving accuracy (%) on Sudoku (left) and zero-shot accuracy (%) on GSM8K (right). † denotes the method trained with self-distillation.

NFE	Model	Easy	Med	Hard
180	MDLM	92.4	68.9	42.1
	Duo	95.8	85.5	61.6
	CANDI	90.4	72.1	34.8
	FLM	93.5	83.4	45.1
	S-FLM	95.8	87.2	45.0
1	FMLM	75.0	42.2	4.9
	FMLM+	67.4	29.9	4.1
	aFMLM (Sync)	75.6	41.8	9.0
	aFMLM (EB)	81.4	53.7	**15.9**
	aFMLM (Alg. order)†	**85.3**	**58.4**	15.4

NFE	Model	Acc.
1024	MDLM	18.0
	Duo	17.2
	CANDI	0.2
	FLM	0.3
	S-FLM	18.0
1	FMLM	0.0
	FMLM+	0.0
	aFMLM (Sync)	0.2
	aFMLM (EB)	**4.1**

Hard and 4.1% on GSM8K, compared with 9.0% and 0.2% for aFMLM (Sync) distilled from the same teacher (Table 1), showing that asynchronous scheduling benefits one-step generation even when the teacher is held fixed. It also outperforms its EB-scheduled aFLM teacher, which achieves 2.0% and 0.0%, respectively, at the same budget. Both advantages persist at 2 and 4 NFE on both tasks (Figure 2), indicating that scheduling and distillation both contribute to accuracy under tight NFE budgets. Beyond these controlled comparisons, aFMLM (EB) outperforms FMLM and FMLM+ across all three Sudoku difficulty levels and on GSM8K at 1 NFE (Table 1). These results show that our method effectively internalizes the state-dependent schedules by distillation.

aFMLM with prescribed schedule. We test whether self-distillation under a known generation schedule can reduce training cost while achieving one-step accuracy comparable to distillation from an EB-scheduled teacher. Using the algorithmic order from Section 5.1, we train aFMLM from scratch via self-distillation without a pretrained teacher. At 1 NFE, aFMLM (Alg. order) achieves 15.4% accuracy on Sudoku-Hard, comparable to the 15.9% of aFMLM (EB) (Table 1). As shown in Table 2, this approach reduces the total training cost from 5.0 to 3.2 GPU hours by eliminating teacher pretraining. These results show that our method effectively distills both prescribed and state-dependent schedules, and that a useful prescribed schedule offers a more efficient route to one-step generation at comparable accuracy.

Table 2: Training cost (GPU hours) on Sudoku-Hard.

	Teacher	Distill	Total
Prescribed	–	3.2h	3.2h
State-dependent	1.7h	3.3h	5.0h

6 Conclusion

We introduced asynchronous flow maps, a framework for distilling ordered generation into one-step flow language models. By treating the tokenwise time of each token as part of the state, we describe asynchronous generation under a scheduler as a single deterministic coupled dynamics, which unifies synchronous, prescribed, and state-dependent schedules. We characterized the flow map of these dynamics and derived a self-distillation objective that jointly learns the evolution of tokens and their tokenwise times, which simplifies to training from scratch when the schedule is prescribed. Empirically, a well-designed scheduler substantially improves the performance of the flow language model, and asynchronous flow maps retain these gains in one or a few steps on Sudoku and GSM8K, without evaluating the scheduler at test time.

AI USE STATEMENT

In this work, we used generative AI tools for finding related literature, drafting and revising the text, and verifying the mathematical derivations and proofs. The authors are fully responsible for the content of the final manuscript and all supplementary materials.

ETHICS STATEMENT

This work uses only publicly available data and involves no human subjects or personal information, and we do not foresee ethical concerns beyond those common to generative language models.

REPRODUCIBILITY STATEMENT

We describe our implementation details in Appendix D, including training and inference hyperparameters. We will publicly release the training checkpoints and the code for full reproduction of our experimental procedure.

REFERENCES

Manan Agarwal, Sheel Shah, Chanhyuk Lee, Jaehoon Yoo, Jerry Huang, Seunghoon Hong, Aditi Raghunathan, Jinwoo Kim, and Nicholas M Boffi. Posterior refinement: Fast language generation via any-order flow maps. *arXiv preprint arXiv:2606.24773*, 2026.

Iskander Azangulov, Kianoosh Ashouritaklimi, Leo Zhang, Simon Vary, and Patrick Rebeschini. Masked language flow models. *arXiv preprint arXiv:2606.27617*, 2026.

Alan Baade, Eric Ryan Chan, Kyle Sargent, Changan Chen, Justin Johnson, Ehsan Adeli, and Li Fei-Fei. Latent forcing: Reordering the diffusion trajectory for pixel-space image generation. *arXiv preprint arXiv:2602.11401*, 2026.

Parikshit Bansal and Sujay Sanghavi. Token time continuous diffusion for language modeling. *arXiv preprint arXiv:2607.14106*, 2026.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11305–11315. IEEE, 2022.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. Langflow: Continuous diffusion rivals discrete in language modeling. *arXiv preprint arXiv:2604.11748*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Justin Deschenaux and Caglar Gulcehre. Language modeling with hyperspherical flows. *arXiv preprint arXiv:2605.11125*, 2026.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

Floor Eijkelboom, Grigory Bartosh, Christian A Naesseth, Max Welling, and Jan-Willem van de Meent. Variational flow matching for graph generation. *Advances in Neural Information Processing Systems*, 37:11735–11764, 2024.

Nima Fathi, Torsten Scholak, and Pierre-André Noël. Unifying autoregressive and diffusion-based sequence generation. *arXiv preprint arXiv:2504.06416*, 2025.

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov, J Zico Kolter, and Andrew Gordon Wilson. From entropy to epiplexity: Rethinking information for computationally bounded intelligence. *arXiv preprint arXiv:2601.03220*, 2026.

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.

Pascal Jutras-Dubé, Patrick Pynadath, Jeremy Lu, Yuan Gao, and Ruqi Zhang. Sticky jump diffusions: A unifying view of masked, continuous, and hybrid diffusion. *arXiv preprint arXiv:2607.10951*, 2026.

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*, 2025.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. Klass: Kl-guided fast inference in masked diffusion models. *Advances in Neural Information Processing Systems*, 38:92267–92301, 2026.

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.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving¿ 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Zhangzhi Peng, Zachary Bezemek, Jarrid Rector-Brooks, Shuibai Zhang, Michael Bronstein, Anru Zhang, Joey Bose, and Alexander Tong. Planner aware path learning in diffusion language models training. In *International Conference on Learning Representations*, volume 2026, pp. 126799–126844, 2026.

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

Patrick Pynadath, Jiaxin Shi, and Ruqi Zhang. Candi: Hybrid discrete-continuous diffusion models. *arXiv preprint arXiv:2510.22510*, 2025.

Bingshuo Qian and Xiang Cheng. Learning when to denoise: Optimizing asynchronous schedules for latent diffusion. *arXiv preprint arXiv:2606.19662*, 2026.

Andrei Rekesh, Jarrid Rector-Brooks, and Cheng-Hao Liu. Inverse-confidence sampling for continuous diffusion language models. In *ICML 2026 Workshop on Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning*, 2026.

Daan Roos, Oscar Davis, Floor Eijkelboom, Michael Bronstein, Max Welling, Ismail Ilkan Ceylan, Luca Ambrogioni, and Jan-Willem van de Meent. Categorical flow maps. *arXiv preprint arXiv:2602.12233*, 2026.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. *Proceedings of machine learning research*, 267: 52584, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Christopher Wewer, Bart Pogodzinski, Bernt Schiele, and Jan Eric Lenssen. Spatial reasoning with denoising models, 2025. URL https://arxiv.org/abs/2502.21075.

Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, Jian Jiao, Juntao Li, Jian Guo, Nan Duan, Weizhu Chen, et al. Ar-diffusion: Auto-regressive diffusion model for text generation. *Advances in Neural Information Processing Systems*, 36:39957–39974, 2023.

A PROOFS

Throughout, we assume that the scheduler and the denoiser are regular enough for (15) to admit unique solutions on $[0, 1)$, so that the associated continuity equations are also uniquely solved, and that prescribed schedules are absolutely continuous. We also assume $\omega_t^i > 0$ whenever $\tau_t^i < 1$, which holds for the schedulers of (27) with $k^i > 0$. Quantities at $t = 1$ are understood as limits $t \uparrow 1$, and diagonal quantities at $t = s$ as limits $t \downarrow s$.

A.1 PROOF OF PROPOSITION 1

Proposition 1. *Consider the objective*

$$\mathcal{L}_{\mathsf{CE}}(\hat{D}) \coloneqq \mathbb{E}_{\tau,\mathbf{x_0},\mathbf{x_1}} \sum_{i=1}^{L} \Big[-\log \hat{D}_\tau^i(I_\tau) \Big]_{\mathbf{y}_i}, \qquad \tau^i \overset{\text{iid}}{\sim} \mathsf{U}[0,1]. \tag{11}$$

Any population minimizer $\hat{D}^\star$ satisfies $\hat{D}_\tau^\star = D_\tau$ almost surely under the training distribution.

Proof. The clean-token distribution conditioned on the interpolant is

$$D_\tau(I_\tau) = \mathbb{E}[\mathbf{x}_1 \mid I_\tau].$$

Conditioning the tokenwise cross-entropy on this input gives

$$\begin{aligned} \mathcal{L}_{\mathsf{CE}}(\hat{D}) &= \mathbb{E}\left[-\sum_{i=1}^{L} \Big\langle D_\tau^i(I_\tau), \log \hat{D}^i(I_\tau, \tau) \Big\rangle \right] \\ &= \mathbb{E}\left[\sum_{i=1}^{L} H(D_\tau^i(I_\tau)) \right] + \mathbb{E}\left[\sum_{i=1}^{L} \mathsf{KL}\Big(D_\tau^i(I_\tau) \,\|\, \hat{D}^i(I_\tau, \tau) \Big) \right]. \end{aligned} \tag{30}$$

The first term does not depend on $\hat{D}$, and the second term is nonnegative and vanishes exactly when $\hat{D}^i(I_\tau, \tau) = D_\tau^i(I_\tau)$ for every i. Thus, $\hat{D}^\star$ minimizes $\mathcal{L}_{\mathsf{CE}}$ if and only if

$$\hat{D}^\star(I_\tau, \tau) = D_\tau(I_\tau) \tag{31}$$

almost surely under the training distribution. □

A.2 PROOF OF PROPOSITION 2

Proposition 2. *There is exactly one continuous nondecreasing path of a scalar time t from 0 to 1 up to a continuous nondecreasing reparameterization. For tokenwise times $\tau \in [0, 1]^L$ with $L \geq 2$, there are infinitely many such paths from $\mathbf{0}$ to $\mathbf{1}$.*

Proof. We parameterize each path by $s \in [0, 1]$.

Scalar time. Let $t(s)$ be a continuous nondecreasing path with

$$t(0) = 0, \qquad t(1) = 1.$$

Then $t(s)$ passes through every value in $[0, 1]$ in order, so it is a reparameterization of $t(s) = s$.

Tokenwise times. It suffices to exhibit one infinite family of such paths that remain distinct up to reparameterization.

For $L \geq 2$ and $a > 0$, let

$$\tau_a(s) = (s^a, s, \ldots, s).$$

Along τ_a, the coordinates satisfy

$$\tau_a^1 = (\tau_a^2)^a,$$

which is unchanged by reparameterization and differs across a. Hence the paths τ_a are infinitely many and distinct up to reparameterization. □

A.3 Proof of Proposition 3

Proposition 3. *Let $\tau(t)$ be a continuous prescribed schedule with $\tau(0) = \mathbf{0}$ and $\tau(1) = \mathbf{1}$. Then the coupled dynamics in* (15) *reduce to*

$$\dot{\mathbf{x}}_t = b_t(\mathbf{x}_t, \tau(t)), \qquad \mathbf{x}_0 \sim p_0, \tag{16}$$

and satisfy $\mathbf{x}_t \sim p_{\tau(t)}$ for every $t \in [0, 1)$, with $p_{\tau(t)} \to p_1$ as $t \uparrow 1$.

Proof. Since ω does not depend on the token states, the tokenwise times follow $\tau_t = \tau(t)$ with $\omega_t = \dot{\tau}(t)$. Substituting into the token component of (15) gives

$$\dot{\mathbf{x}}_t = b_t(\mathbf{x}_t, \tau(t)) = \big[\dot{\tau}(t) \oslash (\mathbf{1} - \tau(t))\big]\big(D_{\tau(t)}(\mathbf{x}_t) - \mathbf{x}_t\big),$$

which is (16). Next, the interpolant $I_{\tau(t)} = (\mathbf{1} - \tau(t))\mathbf{x}_0 + \tau(t)\mathbf{x}_1$ moves with velocity

$$\dot{I}_{\tau(t)} = \dot{\tau}(t)(\mathbf{x}_1 - \mathbf{x}_0).$$

For $\tau^i(t) < 1$, the interpolant gives

$$\mathbf{x}_1^i - \mathbf{x}_0^i = \frac{\mathbf{x}_1^i - I_{\tau(t)}^i}{1 - \tau^i(t)},$$

so its conditional velocity is

$$\mathbb{E}\big[\dot{I}_{\tau(t)}^i \,\big|\, I_{\tau(t)} = \mathbf{x}\big] = \frac{\dot{\tau}^i(t)}{1 - \tau^i(t)}\big(D_{\tau(t)}^i(\mathbf{x}) - \mathbf{x}^i\big) = b_t^i(\mathbf{x}, \tau(t)).$$

Hence $p_{\tau(t)}$ solves the continuity equation

$$\partial_t p_{\tau(t)} + \nabla \cdot \big(p_{\tau(t)}\, b_t(\cdot, \tau(t))\big) = 0, \qquad p_{\tau(0)} = p_0,$$

which is also solved by the law of $\mathbf{x}_t$. By uniqueness, the two coincide, so $\mathbf{x}_t \sim p_{\tau(t)}$. Finally, $\tau(1) = \mathbf{1}$ gives $I_{\tau(1)} = \mathbf{x}_1$, and hence $p_{\tau(t)} \Rightarrow p_1$ as $t \uparrow 1$. □

A.4 Proof of Proposition 4

Proposition 4. *On any interval where* (15) *admits unique solutions, the **asynchronous flow map**, i.e., its solution operator $F_{s,t}(\mathbf{z}_s) = \mathbf{z}_t$, satisfies*

$$F_{s,t}(\mathbf{z}_s) = \mathbf{z}_s + \int_s^t \dot{\mathbf{z}}_r \,\mathrm{d}r, \qquad \textit{and} \qquad F_{s,t} = F_{u,t} \circ F_{s,u}, \qquad s \le u \le t < 1. \tag{17}$$

Proof. Let $(\mathbf{x}_r, \tau_r) = F_{s,r}(\mathbf{z})$, where $\mathbf{z} = (\mathbf{x}, \tau)$. Integrating the coupled dynamics of the token state and tokenwise times from s to t gives

$$F_{s,t}(\mathbf{z}) = \mathbf{z} + \int_s^t \begin{bmatrix} b_r(\mathbf{x}_r, \tau_r) \\ \omega_r(\mathbf{x}_r, \tau_r) \end{bmatrix} \mathrm{d}r.$$

For $s \le u \le t < 1$, both $r \mapsto F_{s,r}(\mathbf{z})$ and $r \mapsto F_{u,r}(F_{s,u}(\mathbf{z}))$ solve the coupled dynamics on $[u, t]$ with the same state $F_{s,u}(\mathbf{z})$ at $r = u$. Uniqueness of solutions therefore gives

$$F_{s,t}(\mathbf{z}) = F_{u,t}(F_{s,u}(\mathbf{z})).$$

□

A.5 Proof of Proposition 5

Proposition 5. *Let $F_{s,t}$ be the flow map of* (15)*. The parameterizations of* (19) *satisfy:*

(i) The flow map is recovered as $F_{s,t}(\mathbf{z}) = (F_{s,t}^x(\mathbf{z}), \tau_{s,t}(\mathbf{z}))$ with

$$F_{s,t}^x(\mathbf{z}) = \mathbf{x} + \frac{\tau_{s,t}(\mathbf{z}) - \tau}{\mathbf{1} - \tau}(\delta_{s,t}(\mathbf{z}) - \mathbf{x}). \tag{20}$$

(ii) On the diagonal $s = t$,

$$\delta_{s,s}(\mathbf{z}) = D_\tau(\mathbf{x}), \qquad \tau_{s,s}(\mathbf{z}) = \tau, \qquad \omega_{s,s}(\mathbf{z}) = \omega_s(\mathbf{x}, \tau). \tag{21}$$

(iii) For $s < u < t$, let $\mathbf{z}_u = F_{s,u}(\mathbf{z})$. The semigroup condition (17) *is equivalent to*

$$\tau_{s,t}(\mathbf{z}) = \tau_{u,t}(\mathbf{z}_u), \qquad \delta_{s,t}(\mathbf{z}) = \gamma\delta_{s,u}(\mathbf{z}) + (\mathbf{1} - \gamma)\delta_{u,t}(\mathbf{z}_u), \tag{22}$$

where

$$\gamma = \frac{(\mathbf{1} - \tau_{u,t}(\mathbf{z}_u)) \odot (\tau_{s,u}(\mathbf{z}) - \tau)}{(\mathbf{1} - \tau_{s,u}(\mathbf{z})) \odot (\tau_{u,t}(\mathbf{z}_u) - \tau)} \in [0, 1]^L. \tag{23}$$

Moreover, $\delta^i_{s,t}(\mathbf{z})$ is simplex-valued whenever D^i is simplex-valued.

Proof. First, let $(\mathbf{x}_r, \tau_r) = F_{s,r}(\mathbf{z})$ with $\mathbf{z} = (\mathbf{x}, \tau)$. By definition, the average velocity and average tokenwise time rate satisfy

$$\begin{aligned}(t - s)v_{s,t}(\mathbf{z}) &= \int_s^t b_r(\mathbf{x}_r, \tau_r)\,\mathrm{d}r = F^x_{s,t}(\mathbf{z}) - \mathbf{x},\\ (t - s)\omega_{s,t}(\mathbf{z}) &= \int_s^t \omega_r(\mathbf{x}_r, \tau_r)\,\mathrm{d}r = \tau_{s,t}(\mathbf{z}) - \tau.\end{aligned} \tag{32}$$

The two-time denoiser therefore satisfies

$$\begin{aligned}\delta^i_{s,t}(\mathbf{z}) &= \mathbf{x}^i + \frac{1 - \tau^i}{\omega^i_{s,t}(\mathbf{z})} v^i_{s,t}(\mathbf{z})\\ &= \mathbf{x}^i + \frac{1 - \tau^i}{\tau^i_{s,t}(\mathbf{z}) - \tau^i}\big(F^{x,i}_{s,t}(\mathbf{z}) - \mathbf{x}^i\big),\end{aligned} \tag{33}$$

using (32).

Rearranging proves (i).

Next, to establish (ii), recall the coupled dynamics along the trajectory:

$$\dot\tau^i_r = \omega^i_r(\mathbf{x}_r, \tau_r), \qquad \dot{\mathbf{x}}^i_r = b^i_r(\mathbf{x}_r, \tau_r) = \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{1 - \tau^i_r}\big(D^i_{\tau_r}(\mathbf{x}_r) - \mathbf{x}^i_r\big).$$

The product rule gives

$$\begin{aligned}\frac{d}{dr}\frac{\mathbf{x}^i_r}{1 - \tau^i_r} &= \frac{b^i_r(\mathbf{x}_r, \tau_r)}{1 - \tau^i_r} + \frac{\omega^i_r(\mathbf{x}_r, \tau_r)\mathbf{x}^i_r}{(1 - \tau^i_r)^2}\\ &= \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{(1 - \tau^i_r)^2} D^i_{\tau_r}(\mathbf{x}_r).\end{aligned} \tag{34}$$

Integrating (34) and using the tokenwise time dynamics,

$$\frac{\mathbf{x}^i_t}{1 - \tau^i_t} - \frac{\mathbf{x}^i}{1 - \tau^i} = \int_s^t \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{(1 - \tau^i_r)^2} D^i_{\tau_r}(\mathbf{x}_r)\,\mathrm{d}r, \tag{35}$$

$$\int_s^t \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{(1 - \tau^i_r)^2}\,\mathrm{d}r = \frac{1}{1 - \tau^i_t} - \frac{1}{1 - \tau^i} = \frac{\tau^i_t - \tau^i}{(1 - \tau^i)(1 - \tau^i_t)}. \tag{36}$$

Substituting the integrated velocity and normalization from (35) and (36) into the denoiser definition (33),

$$\delta^i_{s,t}(\mathbf{z}) = \frac{\displaystyle\int_s^t \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{(1 - \tau^i_r)^2} D^i_{\tau_r}(\mathbf{x}_r)\,\mathrm{d}r}{\displaystyle\int_s^t \frac{\omega^i_r(\mathbf{x}_r, \tau_r)}{(1 - \tau^i_r)^2}\,\mathrm{d}r} \in \Delta^{|V|-1}. \tag{37}$$

The nonnegative weights give the diagonal-limit bound:

$$\|\delta^i_{s,t}(\mathbf{z}) - D^i_\tau(\mathbf{x})\| \le \sup_{r \in [s,t]} \|D^i_{\tau_r}(\mathbf{x}_r) - D^i_\tau(\mathbf{x})\| \xrightarrow[t \downarrow s]{} 0.$$

The tokenwise time displacement in (32) also gives the initial value and the diagonal average tokenwise time rate:

$$\tau_{s,s}(\mathbf{z}) = \tau, \qquad \omega_{s,s}(\mathbf{z}) = \lim_{t\downarrow s} \frac{\tau_{s,t}(\mathbf{z}) - \tau}{t-s} = \lim_{t\downarrow s} \frac{1}{t-s} \int_s^t \omega_r(\mathbf{x}_r, \tau_r)\, \mathrm{d}r = \omega_s(\mathbf{x}, \tau).$$

For (iii), write

$$\mathbf{z}_u = F_{s,u}(\mathbf{z}), \qquad a_i = \tau^i, \qquad b_i = \tau^i_{s,u}(\mathbf{z}), \qquad c_i = \tau^i_{u,t}(\mathbf{z}_u).$$

Applying (i) to each interval gives

$$\begin{aligned} F^{x,i}_{u,t}(\mathbf{z}_u) &= \frac{1-c_i}{1-b_i}\left[\frac{1-b_i}{1-a_i}\mathbf{x}^i + \frac{b_i-a_i}{1-a_i}\delta^i_{s,u}(\mathbf{z})\right] + \frac{c_i-b_i}{1-b_i}\delta^i_{u,t}(\mathbf{z}_u) \\ &= \frac{1-c_i}{1-a_i}\mathbf{x}^i + \frac{c_i-a_i}{1-a_i}\big[\gamma_i\delta^i_{s,u}(\mathbf{z}) + (1-\gamma_i)\delta^i_{u,t}(\mathbf{z}_u)\big], \end{aligned} \tag{38}$$

where the weights are

$$\gamma_i = \frac{(1-c_i)(b_i-a_i)}{(1-b_i)(c_i-a_i)}, \qquad 1-\gamma_i = \frac{(1-a_i)(c_i-b_i)}{(1-b_i)(c_i-a_i)}.$$

The direct and composed maps must first agree in their tokenwise time component, as required by the first identity in (22). Under this equality, subtracting the direct token state transition from (38) yields

$$F^{x,i}_{u,t}(\mathbf{z}_u) - F^{x,i}_{s,t}(\mathbf{z}) = \frac{c_i-a_i}{1-a_i}\big[\gamma_i\delta^i_{s,u}(\mathbf{z}) + (1-\gamma_i)\delta^i_{u,t}(\mathbf{z}_u) - \delta^i_{s,t}(\mathbf{z})\big].$$

Since $c_i > a_i$, the composed and direct token state transitions agree exactly when the second identity in (22) holds. Together with equality of the tokenwise time components, this proves the composition identities (22). □

B Distillation objectives

B.1 Population optimum under state-dependent schedules

We now show that the self-distillation objective of (24) is minimized by the exact two-time denoiser and tokenwise time rate parameterization. We write $\mathcal{L}(\hat{\delta}, \hat{\kappa})$ for the rate-parameterized form of (24) evaluated at $(\hat{\delta}, \hat{\kappa})$, with $\hat{\tau}_{s,t}$ given by (28).

Proposition 6. *Consider* (24) *and* (25) *under fixed training distributions. Let $\lambda > 0$, let $\mathcal{F}$ be the model class, let $(\delta^\star, \kappa^\star)$ be the exact parameterization of the coupled dynamics, and assume $(\delta^\star, \kappa^\star) \in \mathcal{F}$. Then*

$$\mathcal{L}(\delta^\star, \kappa^\star) = \min_{(\hat{\delta},\hat{\kappa})\in\mathcal{F}} \mathcal{L}(\hat{\delta}, \hat{\kappa}) = 0.$$

Conversely, every zero-loss solution satisfies

$$\begin{aligned} \hat{\delta}_{s,s}(\mathbf{z}) &= D_\tau(\mathbf{x}), \\ \hat{\kappa}_{s,s}(\mathbf{z}) &= k_s(\mathbf{x}, \tau), \\ \hat{\delta}_{s,t}(\mathbf{z}) &= \gamma\hat{\delta}_{s,u}(\mathbf{z}) + (\mathbf{1}-\gamma)\hat{\delta}_{u,t}\Big(\hat{F}_{s,u}(\mathbf{z})\Big), \\ \hat{\tau}_{s,t}(\mathbf{z}) &= \hat{\tau}_{u,t}\Big(\hat{F}_{s,u}(\mathbf{z})\Big) \end{aligned}$$

almost surely under the respective training distributions.

Proof. Recall the self-distillation objective:

$$\begin{aligned} \mathcal{L} = \mathbb{E}_{s,\mathbf{z}_s}\Big[\mathsf{KL}(D_{\tau_s}(\mathbf{x}_s) \,\|\, \hat{\delta}_{s,s}(\mathbf{z}_s)) + \lambda\|k_s(\mathbf{x}_s,\tau_s) - \hat{\kappa}_{s,s}(\mathbf{z}_s)\|^2\Big] \\ + \mathbb{E}_{s,u,t,\mathbf{z}_s}\Big[\mathsf{KL}(\bar{\delta}_{s,t} \,\|\, \hat{\delta}_{s,t}(\mathbf{z}_s)) + \lambda\|\bar{\tau}_{s,t} - \hat{\tau}_{s,t}(\mathbf{z}_s)\|^2\Big]. \end{aligned}$$

Its targets are

$$\bar{\tau}_{s,t} = \mathsf{sg}\left[\hat{\tau}_{u,t}(\hat{F}_{s,u}(\mathbf{z}_s))\right],$$
$$\bar{\delta}_{s,t} = \mathsf{sg}\left[\gamma\hat{\delta}_{s,u}(\mathbf{z}_s) + (\mathbf{1}-\gamma)\hat{\delta}_{u,t}(\hat{F}_{s,u}(\mathbf{z}_s))\right].$$

Here $\hat{F}$ is the flow map induced by the candidate model, and γ is the composition weight from (23), evaluated at the candidate's composed tokenwise times. The exact flow satisfies the diagonal and composition identities in Proposition 5, where its diagonal rate is given by (39). Hence

$$\delta^\star_{s,s} = D_\tau, \qquad \kappa^\star_{s,s} = k, \qquad \delta^\star_{s,t} = \bar{\delta}^\star_{s,t}, \qquad \tau^\star_{s,t} = \bar{\tau}^\star_{s,t},$$

where $\tau^\star_{s,t}$ is the tokenwise time induced by $\kappa^\star$ through (28), and $\bar{\delta}^\star_{s,t}$ and $\bar{\tau}^\star_{s,t}$ are the targets of (25) evaluated at $(\delta^\star, \kappa^\star)$. Each term of the objective is therefore zero for the exact flow:

$$0 \le \min_{(\hat{\delta},\hat{\kappa})\in\mathcal{F}} \mathcal{L}(\hat{\delta},\hat{\kappa}) \le \mathcal{L}(\delta^\star,\kappa^\star) = 0.$$

Conversely, all four terms are nonnegative and $\lambda > 0$, so $\mathcal{L} = 0$ if and only if

$$\mathsf{KL}(D_{\tau_s}(\mathbf{x}_s) \,\|\, \hat{\delta}_{s,s}(\mathbf{z}_s)) = 0, \quad \|k_s(\mathbf{x}_s,\tau_s) - \hat{\kappa}_{s,s}(\mathbf{z}_s)\|^2 = 0,$$
$$\mathsf{KL}(\bar{\delta}_{s,t} \,\|\, \hat{\delta}_{s,t}(\mathbf{z}_s)) = 0, \qquad \|\bar{\tau}_{s,t} - \hat{\tau}_{s,t}(\mathbf{z}_s)\|^2 = 0$$

almost surely under the respective training distributions. KL vanishes exactly when its arguments agree. Expanding the two composition targets gives the claimed identities; stop-gradient leaves their values unchanged. □

B.2 Tokenwise time rate reparameterization

Under (27), the tokenwise times satisfy $\partial_r \log(\mathbf{1}-\tau_r) = -k_r/(1-r)$. Integrating over $[s,t]$ shows that (28) is exact when $\kappa_{s,t}$ is the $1/(1-r)$-weighted average of k_r along the trajectory,

$$\kappa_{s,t} = \frac{\int_s^t k_r(\mathbf{x}_r,\tau_r)/(1-r)\,\mathrm{d}r}{\log\frac{1-s}{1-t}}.$$

Here we derive its relation to the diagonal rate $\hat{\omega}_{s,s}$.

Writing the power in (28) as an exponential,

$$\left(\frac{1-t}{1-s}\right)^{\kappa_{s,t}} = \exp\left[\kappa_{s,t}\log\frac{1-t}{1-s}\right],$$

and differentiating with respect to t by the chain and product rules gives

$$\partial_t\left(\frac{1-t}{1-s}\right)^{\kappa_{s,t}} = \left(\frac{1-t}{1-s}\right)^{\kappa_{s,t}} \odot \left[\partial_t\kappa_{s,t}\log\frac{1-t}{1-s} - \frac{\kappa_{s,t}}{1-t}\right].$$

At $t = s$, we have $\frac{1-t}{1-s} = 1$ and $\log\frac{1-t}{1-s} = 0$, so the term involving $\partial_t\kappa_{s,t}$ vanishes and

$$\partial_t\left(\frac{1-t}{1-s}\right)^{\kappa_{s,t}}\Big|_{t=s} = -\frac{\kappa_{s,s}}{1-s}.$$

Substituting this into the derivative of (28),

$$\partial_t\hat{\tau}_{s,t}(\mathbf{z}_s)|_{t=s} = -(\mathbf{1}-\tau_s)\odot\left(-\frac{\kappa_{s,s}}{1-s}\right) = \frac{\mathbf{1}-\tau_s}{1-s}\odot\kappa_{s,s}.$$

Since $\hat{\tau}_{s,t} = \tau_s + (t-s)\,\hat{\omega}_{s,t}$, this derivative is the diagonal rate $\hat{\omega}_{s,s}$. On the other hand, the scheduler of (27) at the same state is

$$\omega_s(\mathbf{x}_s,\tau_s) = \frac{\mathbf{1}-\tau_s}{1-s}\odot k_s(\mathbf{x}_s,\tau_s).$$

Subtracting the two expressions gives

$$\hat{\omega}_{s,s}(\mathbf{z}_s) - \omega_s(\mathbf{x}_s,\tau_s) = \frac{\mathbf{1}-\tau_s}{1-s}\odot\big(\kappa_{s,s}(\mathbf{z}_s) - k_s(\mathbf{x}_s,\tau_s)\big). \tag{39}$$

Since every entry of $(\mathbf{1}-\tau_s)/(1-s)$ is positive for $\tau_s < \mathbf{1}$ and $s < 1$, the diagonal identity $\hat{\omega}_{s,s} = \omega_s$ holds if and only if $\kappa_{s,s} = k$. Moreover, the diagonal term $\lambda\|k - \kappa_{s,s}\|^2$ differs from $\lambda\|\hat{\omega}_{s,s} - \omega_s\|^2$ by a positive position-wise weight given by the square of this factor, so the two terms share the same minimizers.

B.3 PRESCRIBED SCHEDULES

For a prescribed schedule, (29) trains the flow map from scratch with the clean data as the diagonal target instead of a teacher. Since the tokenwise times follow $\tau(t)$ regardless of the token states, the average tokenwise time rate of (18) is $\omega_{s,t} = \frac{\tau(t)-\tau(s)}{t-s}$, and the two-time denoiser of (19) depends only on the token state,

$$\delta^{\tau}_{s,t}(\mathbf{x}) = \mathbf{x} + (t-s)\frac{\mathbf{1}-\tau(s)}{\tau(t)-\tau(s)}\, v^{\tau}_{s,t}(\mathbf{x}), \tag{40}$$

where $v^{\tau}_{s,t}$ is the average velocity of (18) along the schedule. We show that this objective is still minimized by the exact two-time denoiser $\delta^{\tau,\star}$, defined by (40) for the probability flow of (16) with the exact denoiser D_τ. We write $\mathcal{L}_{\mathsf{KL}}(\hat{\delta}^{\tau})$ for this objective evaluated at a candidate $\hat{\delta}^{\tau}$.

For a candidate $\hat{\delta}^{\tau}$, we write $\hat{X}^{\tau}_{s,u}(\mathbf{x}) \coloneqq \mathbf{x} + \frac{\tau(u)-\tau(s)}{\mathbf{1}-\tau(s)}(\hat{\delta}^{\tau}_{s,u}(\mathbf{x}) - \mathbf{x})$ for its induced flow map, following (20).

Proposition 7. *Consider* (25) *and* (29) *under fixed training distributions. Let $\mathcal{F}$ be the model class and $\delta^{\tau,\star}$ the exact two-time denoiser, and assume $\delta^{\tau,\star} \in \mathcal{F}$. Then*

$$\delta^{\tau,\star} \in \arg\min_{\hat{\delta}^{\tau}\in\mathcal{F}} \mathcal{L}_{\mathsf{KL}}(\hat{\delta}^{\tau}).$$

Conversely, every population minimizer satisfies

$$\hat{\delta}^{\tau}_{s,s}(\mathbf{x}) = D_{\tau(s)}(\mathbf{x}),$$
$$\hat{\delta}^{\tau}_{s,t}(\mathbf{x}) = \gamma\hat{\delta}^{\tau}_{s,u}(\mathbf{x}) + (\mathbf{1}-\gamma)\hat{\delta}^{\tau}_{u,t}\Big(\hat{X}^{\tau}_{s,u}(\mathbf{x})\Big)$$

almost surely under the respective training distributions.

Proof. Recall that the objective combines KL divergences against the clean data and the composition target:

$$\mathcal{L}_{\mathsf{KL}} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}(\mathbf{x}_1 \,\|\, \hat{\delta}^{\tau}_{t,t}(I_{\tau(t)}))\right] + \mathbb{E}_{s,u,t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}(\bar{\delta}^{\tau}_{s,t} \,\|\, \hat{\delta}^{\tau}_{s,t}(I_{\tau(s)}))\right],$$
$$\bar{\delta}^{\tau}_{s,t} = \mathsf{sg}\left[\gamma\hat{\delta}^{\tau}_{s,u}(I_{\tau(s)}) + (\mathbf{1}-\gamma)\hat{\delta}^{\tau}_{u,t}(\hat{X}^{\tau}_{s,u}(I_{\tau(s)}))\right].$$

Here $\hat{X}^{\tau}$ is the flow map induced by the candidate denoiser, and γ is the composition weight from (23) evaluated at $\tau(s), \tau(u), \tau(t)$. Since $\mathbf{x}_1^i$ is one-hot, the first KL term equals the tokenwise cross-entropy, and applying the cross-entropy decomposition proved in Appendix A.1,

$$C^{\tau} \coloneqq \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\sum_{i=1}^{L} H(D^{i}_{\tau(t)}(I_{\tau(t)}))\right], \tag{41}$$

$$\mathcal{L}_{\mathsf{KL}} - C^{\tau} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}\Big(D_{\tau(t)}(I_{\tau(t)}) \,\|\, \hat{\delta}^{\tau}_{t,t}(I_{\tau(t)})\Big)\right] + \mathbb{E}_{s,u,t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathsf{KL}\Big(\bar{\delta}^{\tau}_{s,t} \,\|\, \hat{\delta}^{\tau}_{s,t}(I_{\tau(s)})\Big)\right] \geq 0. \tag{42}$$

Here KL sums over token positions. By Proposition 5 applied to the probability flow of (16), the exact two-time denoiser satisfies the diagonal and composition identities with $\tau_{s,t} = \tau(t)$, so both KL terms vanish:

$$\mathcal{L}_{\mathsf{KL}}(\delta^{\tau,\star}) = C^{\tau} = \min_{\hat{\delta}^{\tau}\in\mathcal{F}} \mathcal{L}_{\mathsf{KL}}(\hat{\delta}^{\tau}).$$

Conversely, by (42), $\mathcal{L}_{\mathsf{KL}} = C^{\tau}$ if and only if

$$\hat{\delta}^{\tau}_{t,t}(I_{\tau(t)}) = D_{\tau(t)}(I_{\tau(t)}), \qquad \hat{\delta}^{\tau}_{s,t}(I_{\tau(s)}) = \bar{\delta}^{\tau}_{s,t}$$

almost surely under the respective training distributions. Expanding the composition target gives the claimed identity. Stop-gradient does not change the target value. □

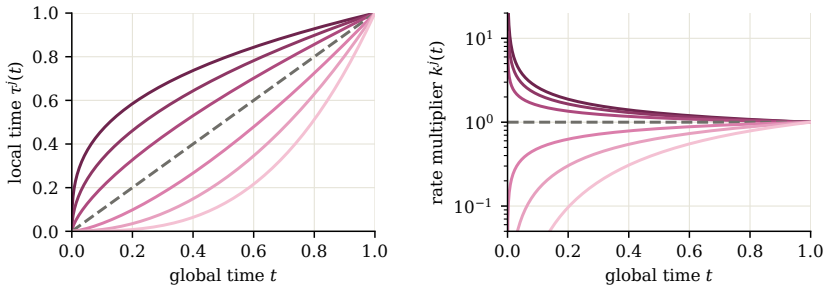

Figure 3: Power schedules $\tau^j(t) = t^{\alpha_j}$ with $\beta = \log 3$ (left) and their rate multipliers k_t^j (right). Darker curves denote higher-priority positions, with exponents ranging from $\alpha_j = 1/3$ (darkest) to $\alpha_j = 3$ (lightest), and the dashed line denotes the synchronous schedule.

C Construction of schedulers

All schedulers in our experiments take the common form of (27),

$$\omega_t^i(\mathbf{x}, \tau) = \frac{1 - \tau^i}{1 - t} k_t^i(\mathbf{x}, \tau), \qquad t < 1.$$

The schedulers differ in what the rate multiplier k^i depends on. Let

$$p_i = D_\tau^i(\mathbf{x}), \qquad c_i = \|p_i\|_\infty, \qquad h_i = H(p_i) \tag{43}$$

denote the predicted clean-token distribution, its confidence, and its entropy, respectively. All position indices and statistics below refer to generated tokens, and fixed context tokens remain unchanged.

Power schedules. Several of the schedulers below use the power schedule

$$\tau^j(t) = t^{\alpha_j} \tag{44}$$

(Figure 3). Its rate $\omega_t^j = \alpha_j t^{\alpha_j - 1}$ takes the common form with a rate multiplier that depends only on t,

$$k_t^j = \omega_t^j \frac{1 - t}{1 - \tau^j(t)} = \alpha_j t^{\alpha_j - 1} \frac{1 - t}{1 - t^{\alpha_j}}, \tag{45}$$

which tends to one as $t \uparrow 1$. Given an order over the generated positions, let $\rho_j \in [0, 1]$ denote the normalized rank of position j, with zero denoting the highest priority, and set

$$\alpha_j = \exp\big[\beta\,(2\rho_j - 1)\big], \tag{46}$$

so that the exponents range from $e^{-\beta}$ to e^{β} and positions with higher priority are denoised faster. Except for the synchronous schedule, we use $\beta = \log 3$, so that the exponents range from $1/3$ to 3. We use the following power schedules.

- **Synchronous.** Setting $\beta = 0$ gives $k^j = 1$ and $\tau^j(t) = t$, which recovers the synchronous schedule of Lee et al. (2026).
- **Left-to-right.** ρ_j is the normalized position of token j among the generated positions.
- **Algorithmic order for Sudoku.** $\rho_j = r_{\mathbf{c}}(j)/R_{\mathbf{c}}$, where $r_{\mathbf{c}}(j)$ is the round in which cell j is resolved and $R_{\mathbf{c}}$ is the last round (Appendix D.2). Cells resolved in the same round share the same exponent.
- **Initial confidence.** Similar to Bansal & Sanghavi (2026), ρ_j is the rank of position j by decreasing confidence of the initial prediction $D_{\mathbf{0}}(\mathbf{x}_0)$, kept fixed during generation. k thus depends on the token states only through the initial state $\mathbf{x}_0$.

The following two schedulers instead set k^i from the current predictions at every step.

Top-k confidence scheduler. Following MaskGIT (Chang et al., 2022), we advance the most confident positions faster, and increase their fraction over time with the cosine schedule $f(t) =$

$1 - \cos(\pi t/2)$. Let $\widetilde{c}_i$ denote the confidence c_i standardized over the generated positions, and let $q(t)$ be its quantile at level $1 - f(t)$, so that a fraction $f(t)$ of positions have $\widetilde{c}_i > q(t)$. We set

$$k^i = k_{\min} + A\,\sigma\left(\frac{\widetilde{c}_i - q(t)}{T}\right), \tag{47}$$

where σ is the sigmoid, $k_{\min} > 0$ is the rate multiplier of the remaining positions, $A > 0$ is the additional rate for the selected ones, and $T > 0$ is a temperature that controls how sharply the rate changes around the threshold.

Entropy-bounded scheduler. Following EB-Sampler (Ben-Hamu et al., 2025), we order token positions by decreasing confidence or increasing entropy. Let π_j denote the j-th position in this order. For the first j positions, we sum their entropies and subtract the largest:

$$B_j = \sum_{\ell=1}^{j} h_{\pi_\ell} - \max_{1\le\ell\le j} h_{\pi_\ell}. \tag{48}$$

We assign larger rate multipliers to positions before B_j exceeds a threshold γ, using a sigmoid to decrease the multiplier gradually:

$$k^{\pi_j} = k_{\min} + A\,\sigma\left(\frac{\gamma - B_j}{T}\right). \tag{49}$$

Here $A > 0$ controls the additional rate multiplier, and $T > 0$ controls how sharply it decreases around $\gamma \ge 0$.

D Experiment Details

D.1 Training details

Architecture and hyperparameters. All models use a Diffusion Transformer (Peebles & Xie, 2023) backbone with adaLN time conditioning. All models on the same task share the architecture, and the tokenwise time $\tau \in [0,1]^L$ is embedded per position for the diffusion-forcing teacher and the asynchronous flow maps. Per-task hyperparameters are listed in Table 3. We train all models with AdamW (Loshchilov & Hutter, 2017) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and no weight decay. The learning rate is linearly warmed up to 3×10^{-4} over 2,500 steps and kept constant afterwards, and gradients are clipped to norm 1.0. We use bfloat16 mixed precision and keep an exponential moving average of the weights with decay 0.9999, which is used for evaluation. All models are trained on NVIDIA A100 and H100 GPUs.

Teacher training. The synchronous FLM teacher is trained with the cross-entropy loss of (3) with $t \sim \mathsf{U}[0,1]$ shared by all positions, and the diffusion-forcing teacher is trained with (11). Both teachers are trained for the number of steps in Table 3.

Distillation. The state-dependent asynchronous flow maps are initialized from the diffusion-forcing teacher, while the prescribed asynchronous flow maps are self-distilled from scratch. For the final results of the state-dependent asynchronous flow maps, we use the entropy-bounded scheduler for both Sudoku and GSM8K. Following Lee et al. (2026), we sample three times $s < u < t$ by first sampling $d = t - s \sim \mathsf{U}[0,1]$, then sampling $s \sim \mathsf{U}[0, 1-d]$, and setting $u = (s+t)/2$. For the state-dependent flow maps, the state at time s is obtained by rolling out the dynamics from time 0 to s. We run this rollout curriculum for five rounds, where the first round rolls out the teacher with 32 steps and each later round rolls out the student with half as many steps, down to 2 steps. The final round with 2 rollout steps continues for the remaining training steps. We set the tokenwise time loss weight in (24) to $\lambda = 1.0$.

Scheduler hyperparameters. For top-k confidence scheduling, we use $(T, A) = (0.2, 12)$ on all Sudoku difficulty levels and $(0.5, 48)$ on GSM8K, with $k_{\min} = 0.1$. For entropy-bounded scheduling, we use $(\gamma, T, A) = (0.01, 0.001, 12)$ on all Sudoku difficulty levels and $(3, 0.3, 24)$ on GSM8K, with $k_{\min} = 0.1$ throughout. For the prescribed power schedules reported here, we use $\beta = \log 3$, giving exponents in $[1/3, 3]$, except for the synchronous schedule, which uses $\beta = 0$. For initial confidence scheduling, we rank the positions by the confidence of the initial prediction and assign the power schedule with $\beta = \log 3$ by this rank.

Table 3: **Per-task hyperparameters.** All models on the same task share the architecture and optimizer, and differ only in the training and inference schedules.

	TinyGSM / GSM8K	Sudoku
Architecture (DiT)		
Hidden size	768	512
adaLN cond. dim	128	128
Layers	12	8
Attention heads	12	8
FFN multiplier	4	4
Dropout	0.1	0.1
Sequence length L	512	180
Vocabulary $\lvert V \rvert$	49152 (SmolLM-135M)	12
Parameters (approx.)	168M	29M
Teacher training		
Optimizer	AdamW (β_1=0.9, β_2=0.999, ϵ=10^{-8})	
Weight decay	0	
Gradient clipping	1.0	
Schedule	Constant w/ linear warmup	
Warmup steps	2 500	
Peak LR	3×10^{-4}	3×10^{-4}
Total steps	250,000	20,000
Global batch size	512	256
Precision	bfloat16 mixed	
EMA decay	0.9999	
Distillation		
Total steps	100,000	20,000
Peak LR	3×10^{-4}	3×10^{-4}
Global batch size	512	256
Training steps per curriculum round	2,000	2,000
Initial rollout steps	32	32

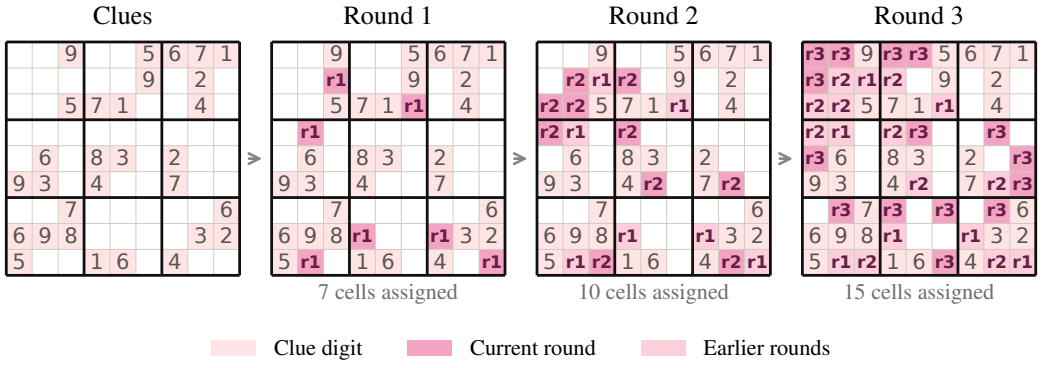

Figure 4: Algorithmic ordering for a hard Sudoku puzzle. Plain digits are the given clues; labels `r1`–`r3` indicate the round in which each empty cell is resolved. Cells resolved in the current round are shaded in darker magenta, while previously assigned rounds remain visible. Cells still blank after round 3 are resolved in later rounds (not shown).

Evaluation. All models are sampled on a uniform grid of the global time t with the given NFE, where the teachers integrate the ODE and the flow maps jump between adjacent grid points, and the tokenwise times evolve according to the scheduler. We evaluate on the full 1,319 GSM8K test questions and 2,000 Sudoku puzzles for each difficulty, and report exact-match accuracy.

D.2 ALGORITHMIC ORDERING FOR SUDOKU

Starting from the given clues $\mathbf{c}$, we compute the valid digits for each empty cell under the row, column, and box constraints. At each round, we identify cells with only one valid digit and digits with only one valid location within a row, column, or box. We fill all identified cells simultaneously and recompute the candidates, repeating until no further cells can be resolved. Figure 4 illustrates the first three rounds on a hard Sudoku puzzle.

Let $r_{\mathbf{c}}(j)$ denote the round in which cell j is first resolved, and let $R_{\mathbf{c}}$ be the last such round, with $R_{\mathbf{c}} = 1$ if no cells are resolved. We assign unresolved cells $r_{\mathbf{c}}(j) = R_{\mathbf{c}}$.