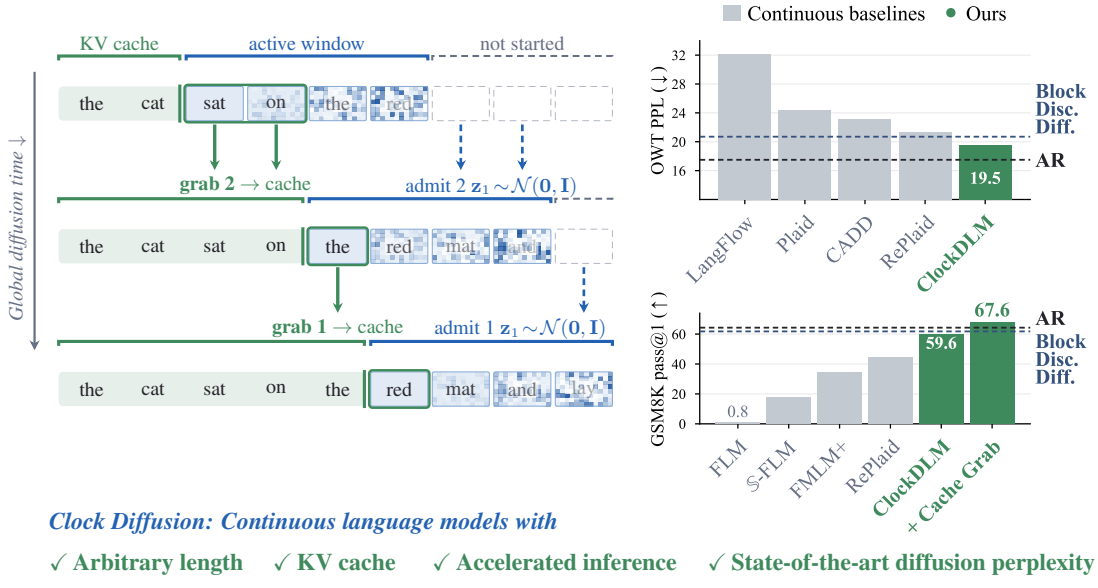
CLOCK DIFFUSION: EFFICIENT SEMI-AUTOREGRESSIVE CONTINUOUS DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent works on continuous diffusion for discrete data have demonstrated performance on par with comparable discrete diffusion models. However, these continuous counterparts lack key features that are essential to practical use as language models, namely variable-length generation and support for a key-value cache, and they still lag behind the frontier of autoregressive and discrete diffusion quality. In this work, we address these limitations. We do so by introducing a model parameterization that uses position-dependent noise schedules to define semi-autoregressive (SAR) continuous diffusion language models (DLMs). Together with efficient training and sampling algorithms, we call this framework **Clock Diffusion**, and we present two special cases of our method: block and sliding window generation. We then define **ClockDLMs**, a family of Gaussian DLMs based on sliding window Clock Diffusion that attain state-of-the-art diffusion likelihood bounds on OpenWebText, even beating the performant block SAR discrete diffusion models. ClockDLMs trained on TinyGSM also substantially outperform continuous baselines on the GSM8K benchmark and match and exceed comparable SAR discrete diffusion models. Finally, building on our parameterization, we propose more efficient samplers that we dub **Cache Grab**, which adapt techniques from accelerated inference in discrete diffusion, such as committing tokens whose probabilities exceed a confidence threshold and self-speculative decoding, further improving our models' quality and efficiency.



Clock Diffusion: Continuous language models with

✓ Arbitrary length ✓ KV cache ✓ Accelerated inference ✓ State-of-the-art diffusion perplexity

Figure 1: (*Left*) **Clock Diffusion** formulates semi-autoregressive continuous diffusion for discrete data, unlocking key drivers of practical LMs. (*Right*) **ClockDLMs** achieve SoTA diffusion language modeling, beating block discrete diffusion and setting a new standard for continuous LMs. Our **Cache Grab** sampler further pushes the frontier, exceeding AR accuracy on GSM8K.

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.

1 INTRODUCTION

Efforts to adapt diffusion models (Sohl-Dickstein et al., 2015; Song et al., 2019; Ho et al., 2020; Song et al., 2020) to discrete domains, such as text, can be split into two streams of work. The first relies on embedding the data in a continuous representation and then applying standard continuous diffusion on this space (Li et al., 2022; Gulrajani et al., 2023). The second and historically more successful approach thus far has been to directly define the corruption and denoising processes on the discrete representations (Austin et al., 2021; Campbell et al., 2022; Lou et al., 2024; Sahoo et al., 2024a; Schiff et al., 2024). A recent resurgence of continuous models has challenged this paradigm (Chen et al., 2026; Davis et al., 2026; Lee et al., 2026; Potaptchik et al., 2026; Roos et al., 2026; Yang et al., 2026), showing competitive performance with full sequence discrete diffusion. However, these latest full-sequence continuous models cannot be used in practice, as they do not support variable-length generation with KV caching. Moreover, their performance lags behind autoregressive (AR) language models (LMs) and semi-autoregressive (SAR) discrete diffusion.

In this work, we therefore present SAR continuous diffusion language models (DLMs), unlocking arbitrary length generation and KV caching for continuous models while substantially improving quality. We propose a framework consisting of variable noise schedules along the sequence (Ruhe et al., 2024) and efficient training and sampling algorithms that we call **Clock Diffusion**, owing to our use of position-dependent local clocks that reparameterize the global time of the diffusion process. We use our parameterization to formulate block and sliding window SAR generation.

Combining Gaussian diffusion over token embeddings with sliding window Clock Diffusion, we introduce ClockDLMs. We train and evaluate these models on widely used language datasets. Our model is the first continuous DLM to match block discrete diffusion on perplexity and sets a new state-of-the-art among continuous models.

For the sliding window model, our sampling algorithm enables KV cache updates at every iteration. Building on this, we propose accelerated inference methods that we dub **Cache Grab** samplers, which adapt techniques from discrete diffusion, such as committing tokens with probabilities above a confidence threshold (Wu et al., 2025) and self-speculative decoding (Fu et al., 2026).

Contributions In summary, our contributions are as follows: 1) We introduce Clock Diffusion, a framework for variable-length generation and KV caching for continuous DLMs that consists of a probabilistic model and efficient training and sampling algorithms. 2) We define ClockDLMs, a new family of Gaussian DLMs based on sliding window Clock Diffusion. 3) We develop samplers, which accelerate inference with ClockDLMs by adapting successful methods from discrete diffusion. We name these Cache Grab samplers. 4) ClockDLMs achieve SoTA among all continuous DLMs on language modeling (as measured by PPL on OWT), even beating discrete block DLMs. When combined with our Cache Grab samplers, we also achieve the highest GSM8K accuracy among comparably sized models trained on TinyGSM.

2 BACKGROUND

Diffusion Processes Diffusion uses a forward corruption process that progressively corrupts samples $\mathbf{x} \sim q_{\text{data}}$ through a Markov chain of time-indexed latent variables $\mathbf{z}_t$, for $t \in [0, 1]$, so that the terminal distribution approximates a tractable prior q_{prior}. The *forward* corruption process $q_{t|s}(\mathbf{z}_t \mid \mathbf{z}_s)$, where for $s < t$, latent variables become progressively noisier, can be applied recursively to yield tractable noising kernels $q_t(\mathbf{z}_t \mid \mathbf{x})$, which in turn can be used to form tractable *reverse* posteriors $q_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x}) = q_{t|s}(\mathbf{z}_t \mid \mathbf{z}_s)\frac{q_s(\mathbf{z}_s|\mathbf{x})}{q_t(\mathbf{z}_t|\mathbf{x})}$. Diffusion generative models are defined by learned reverse distributions, which are commonly parameterized as $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t) := q_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \mathbf{x}_\theta)$, where $\mathbf{x}_\theta$ is a neural network prediction of the clean signal $\mathbf{x}$ given the noisy input $\mathbf{z}_t$. For multivariate data $\mathbf{x}, \mathbf{z}_t \in \mathbb{R}^d$, both the forward and reverse processes are applied independently across dimensions. For sequence data, we model $\mathbf{x}^{1:L}, \mathbf{z}_t^{1:L} \in \mathbb{R}^{L\times d}$, where superscripts denote the sequence length dimension, and we apply the diffusion processes independently along this dimension as well.

Continuous Diffusion for Text To enable Gaussian corruption for discrete data, we embed tokens into a continuous space where the diffusion process is applied. Let $\mathbf{x}^\ell \in \{0, 1\}^V \cap \Delta^V$ denote

a one-hot vector, where V is the vocabulary size and Δ^V is the simplex over the vocabulary. An embedder $E : \Delta^V \to \mathbb{R}^d$ produces embeddings $\mathbf{e}^\ell := E[\mathbf{x}^\ell] \in \mathbb{R}^d$. We denote a sequence of L embeddings as $\mathbf{e}^{1:L} := E[\mathbf{x}^{1:L}] \in \mathbb{R}^{L\times d}$. The diffusion process is then defined as follows:

$$q(\mathbf{z}_t^\ell \mid \mathbf{x}^\ell) = \mathcal{N}(\alpha_t \mathbf{e}^\ell, \sigma_t^2 \mathbf{I}) \quad \text{and} \quad q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) = \mathcal{N}\left(\tfrac{\alpha_{t|s}\sigma_s^2}{\sigma_t^2}\mathbf{z}_t^\ell + \tfrac{\alpha_s \sigma_{t|s}^2}{\sigma_t^2}\mathbf{e}^\ell, \tfrac{\sigma_{t|s}^2\sigma_s^2}{\sigma_t^2}\mathbf{I}\right), \tag{1}$$

where $\alpha_{t|s} := \alpha_t/\alpha_s, \sigma_{t|s}^2 := \sigma_t^2 - \alpha_{t|s}^2\sigma_s^2$, and $\alpha_t := \alpha(t)$ and $\sigma_t := \sigma(t)$ are monotonically decreasing and increasing functions in t, respectively, that dictate the amount of signal and noise, respectively, present in a given latent variable. For the learned reverse process, often, we let $\mathbf{x}_\theta^\ell \in \Delta^V$ be the softmax output of a neural network that predicts logits over the vocabulary at position ℓ and our parameterized prediction of the clean token embeddings is $\mathbf{e}_\theta^{1:L} := E[\mathbf{x}_\theta^{1:L}]$.

Previous continuous LMs can be categorized by the embedder E that they use, with some notable examples being: *One-hot embedding*: $E := \mathrm{Id} : \Delta^V \to \mathbb{R}^V$ (Lee et al., 2026; Potaptchik et al., 2026; Roos et al., 2026), *Token embeddings*: E is the embedding matrix $\in \mathbb{R}^{V\times d}$ (Li et al., 2022; Dieleman et al., 2022; Gulrajani et al., 2023; Chen et al., 2026; Yang et al., 2026), and *Contextual embeddings*: E is a pre-trained encoder (Hu et al., 2026).

Semi-Autoregressive Discrete Diffusion In the context of discrete diffusion, Arriola et al. (2025a) decompose the distribution over sequences into sequential blocks and perform diffusion over each block. This SAR approach tightens the likelihood bound, improves sample quality, and most notably unlocks KV cache sampling when using Transformer architectures (Vaswani et al., 2017).

3 Clock Diffusion

In this section, we present our framework, which consists of a position-dependent noising parameterization (Section 3.1), a KV cache sampling algorithm (Section 3.2), and efficient training (Section 3.3). In Section 3.4, we provide block and sliding window instantiations of this framework, and in Section 3.5, we define ClockDLMs, a family of Gaussian DLMs built on our sliding window parameterization. Finally, in Section 3.6, we develop efficient samplers for ClockDLMs.

3.1 Parameterization

Forward Process We build on previous works that proposed position-dependent noise schedules in the context of continuous data (Chen et al., 2024; Ruhe et al., 2024), such as video frame prediction or robotic motion planning, and adapt these methods to language. Starting from a continuous diffusion for discrete data noising process, as described in Section 2, the key insight is that we can have each position $\ell \in \{1, \dots, L\}$ convert global diffusion time t via a local clock $u^\ell : [0,1] \to [0,1]$, which is non-decreasing and must satisfy $u^\ell(0) = 0$ and $u^\ell(1) = 1$. This defines per-position noising schedules: $q_{t|s}(\mathbf{z}_t^{1:L} \mid \mathbf{z}_s^{1:L}) = \prod_{\ell=1}^L q_{u^\ell(t)|u^\ell(s)}(\mathbf{z}_t^\ell \mid \mathbf{z}_s^\ell)$. Importantly, we also impose that for $u^\ell(t) = u^\ell(s) = u$, i.e., for a global diffusion time progression that leads to no change in a token's local clock, the noising kernel is the Dirac measure:

$$q_{u|u}(\mathbf{z}_t^\ell \mid \mathbf{z}_s^\ell) := \delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell) \to \textit{positions where local time has not advanced remain fixed}. \tag{2}$$

Reverse Process In the denoising direction, using local clocks, we create two non-overlapping sets:

$$\text{`}\textit{awake}\text{'}\ \ \mathcal{A}_{s|t} := \{\ell \mid u^\ell(s) < u^\ell(t)\}, \quad \text{`}\textit{snoozed}\text{'}\ \ \mathcal{Z}_{s|t} := \{\ell \mid u^\ell(s) = u^\ell(t)\} = (\mathcal{A}_{s|t})^c. \tag{3}$$

Awake positions are actively being denoised, as they undergo a local clock change. *Snoozed* positions correspond to latent variable components that can be safely ignored, as they consist of tokens that either have not begun $u^\ell(s) = u^\ell(t) = 1$ or have completed denoising $u^\ell(s) = u^\ell(t) = 0$.

The kernel from (2) lets us derive a Dirac measure for the posterior of tokens in the *snoozed* set:

$$q_{u^\ell(s)|u^\ell(t)}(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) = q_{u|u}(\mathbf{z}_t^\ell \mid \mathbf{z}_s^\ell)\tfrac{q_u(\mathbf{z}_s^\ell|\mathbf{x}^\ell)}{q_u(\mathbf{z}_t^\ell|\mathbf{x}^\ell)} = \delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell), \quad \text{if } \ell \in \mathcal{Z}_{s|t}, \tag{4}$$

where in the last equality we used the fact that for any function we have $\delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell) f(\mathbf{z}_s^\ell) = \delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell) f(\mathbf{z}_t^\ell)$ applied to $f(\mathbf{z}_s^\ell) = q_u(\mathbf{z}_s^\ell \mid \mathbf{x}^\ell)/q_u(\mathbf{z}_t^\ell \mid \mathbf{x}^\ell)$. We can then characterize the full reverse posterior as factorizing over awake and snoozed positions:

$$q_{s|t}(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L}) = \textstyle\prod_{\ell\in\mathcal{A}_{s|t}} q_{u^\ell(s)|u^\ell(t)}(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) \prod_{\ell\in\mathcal{Z}_{s|t}} \delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell). \tag{5}$$

Generative Model For the learned process, define the set of completed positions at global time t as $\mathcal{C}_t := \{\ell \mid u^\ell(t) = 0\}$ and the newly completed positions for times $s < t$ as $\mathcal{C}_{s|t} := \mathcal{C}_s \setminus \mathcal{C}_t$. Let $\mathbf{z}_{0:1}^{1:L} := (\mathbf{z}_{t(i)}^{1:L})_{i=0}^{T-1}$ denote the latent trajectory. Then we posit a generative model whose joint distribution over data and latent variable sequences factorizes as follows:

$$\begin{aligned} p_\theta(\mathbf{x}^{1:L}, \mathbf{z}_{0:1}^{1:L}) &= p_\theta(\mathbf{z}_1^{1:L}) \prod_{i=1}^{T-1} p_\theta\big(\mathbf{z}_s^{1:L}, \mathbf{x}^{\mathcal{C}_{s|t}} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t}\big) \\ &:= p_\theta(\mathbf{z}_1^{1:L}) \prod_{i=1}^{T-1} p_\theta\big(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t}\big)\, p_\theta\big(\mathbf{x}^{\mathcal{C}_{s|t}} \mid \mathbf{z}_s^{1:L}, \mathbf{x}^{\mathcal{C}_t}\big)\,. \end{aligned} \tag{6}$$

This factorization dictates that at every timestep, we model latent sequence denoising $\mathbf{z}_t^{1:L} \to \mathbf{z}_s^{1:L}$ and then account for updates to the clean sequence given these changes to the denoised latent. We define the latent-transition factor in (6) so that it factorizes over snoozed and awake positions:

$$p_\theta\big(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t}\big) := p_\theta(\mathbf{z}_s^{\mathcal{Z}_{s|t}} \mid \mathbf{z}_t^{\mathcal{Z}_{s|t}}) p_\theta(\mathbf{z}_s^{\mathcal{A}_{s|t}} \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t}). \tag{7}$$

Following the true reverse posterior, we can ignore snoozed positions by defining for any $\ell \in \mathcal{Z}_{s|t}$: $p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell) := \delta(\mathbf{z}_t^\ell - \mathbf{z}_s^\ell)$. We then need only account for the awake positions, which depend on their previous state at time t and any already revealed tokens $\mathbf{x}^{\mathcal{C}_t}$.

Finally, to model updates to the data sequence, we output distributions over the clean data given the updated awake positions and already revealed tokens:

$$p_\theta(\mathbf{x}^{\mathcal{C}_{s|t}} \mid \mathbf{z}_s^{1:L}, \mathbf{x}^{\mathcal{C}_t}) := p_\theta(\mathbf{x}^{\mathcal{C}_{s|t}} \mid \mathbf{z}_s^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t}) := \textstyle\prod_{\ell \in \mathcal{C}_{s|t}} \mathrm{Cat}(\mathbf{x}^\ell; \mathbf{x}_\theta^\ell(\mathbf{z}_s^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})), \tag{8}$$

where $\mathrm{Cat}(\cdot; p)$ denotes a categorical distribution with probability vector $p \in \Delta^V$.

Semi-Autoregressive Local Clocks In our work, we study local clocks for which every position runs the same schedule but is offset by a per-position *phase* $\varphi(\ell)$:

$$u^\ell(t) := \mathrm{clip}_{[0,1]}\big(r\, t - \varphi(\ell)\big), \qquad r := \max_{\ell' \in \{1,\dots,L\}} \varphi(\ell') + 1, \tag{9}$$

where $\varphi : \{1, \dots, L\} \to \mathbb{R}_{\geq 0}$ and $\mathrm{clip}_{[0,1]}(\cdot)$ clips its argument to $[0, 1]$. As shown in Section 3.4, different choices of φ lead to different instantiations of Clock Diffusion. Intuitively, $\varphi(\ell)$ is the '*head start*' of position ℓ. Each position experiences rescaled time $t \to r\, t$ and is awake precisely for all $t \in [0, 1]$ such that $\varphi(\ell) < r\, t < \varphi(\ell) + 1$, i.e., every position denoises over a window that lasts $1/r$ units of global diffusion time, and $\varphi(\ell)$ sets where that window falls. This parameterization induces SAR decoding: positions with a larger phase finish denoising earlier.

Since all token windows are of equal duration, for any two positions ℓ, ℓ' the offset between their windows is determined by $G_\varphi(\ell, \ell') := \varphi(\ell') - \varphi(\ell)$. When $G_\varphi \geq 1$ windows are disjoint, so ℓ' finishes denoising before or just as ℓ begins; when $G_\varphi = 0$ the two positions denoise in lockstep; and when $0 < G_\varphi < 1$ they overlap, so ℓ' leads ℓ but the two are denoised together for part of their trajectories. Negative G_φ gives the same three cases with the roles of ℓ and ℓ' exchanged.

Variational Bound Clock Diffusion admits a principled likelihood bound (see Appendix A):

$$\begin{aligned} \mathcal{L}(\theta) := \underbrace{\mathrm{KL}[q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L}) || p_\theta(\mathbf{z}_1^{1:L})]}_{\mathcal{L}_{\text{prior}}} + \underbrace{\textstyle\sum_{i=1}^{T-1} \mathbb{E}_{q(\mathbf{z}_s^{\mathcal{A}_{s|t}} | \mathbf{x}^{1:L})} \sum_{\ell \in \mathcal{C}_{s|t}} - \log p_\theta(\mathbf{x}^\ell \mid \mathbf{z}_s^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})}_{\mathcal{L}_{\text{recons}}} \\ + \underbrace{\textstyle\sum_{i=1}^{T-1} \mathbb{E}_{q(\mathbf{z}_t^{1:L} | \mathbf{x}^{1:L})} \sum_{\ell \in \mathcal{A}_{s|t}} \mathrm{KL}[q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) || p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})]}_{\mathcal{L}_{\text{diffusion}}}. \end{aligned} \tag{10}$$

3.2 Key-Value Cache Sampling

At any global time t, the local clocks split the sequence into disjoint sets: positions in $\mathcal{C}_t$ have finished denoising, awake positions are being actively denoised, and the rest have not yet begun. Positions in $\mathcal{C}_t$ never change again and therefore their KV entries are written once, when the position completes, and reused unchanged thereafter. Every awake clock advances at rate r, so evaluating t at T equally spaced values moves each awake clock by r/T per model call. Giving every position a budget of N reverse updates therefore fixes the number of model calls at $T := r\, N$.

Algorithm 1 is the resulting sampler. At each step, we advance the entire set $\mathcal{A}_{s|t}$ with a single call to $\textsc{Denoiser}(\mathbf{z}_t^{\mathcal{A}_{s|t}}, u^{\mathcal{A}_{s|t}}(t), \mathbf{KV})$. Positions whose clocks just reached zero are collected into $\mathcal{C}_{s|t}$, decoded via $\arg\max$, and appended to the cache with a clean-token model forward, written as $\textsc{Cache}(\mathbf{x}^{\mathcal{C}_{s|t}}, \mathbf{KV})$. In practice, we avoid the separate model forward $\textsc{Cache}$ call by fusing this with the subsequent denoiser step, requiring a custom attention mask, depicted in Appendix D.

Algorithm 1 Clock Diffusion Sampling

Require: prompt, length L, per-position budget N, clocks u^ℓ, prior q_{prior}, $\textsc{Denoiser}$, sampling routine $\textsc{Sampler}$

$T \leftarrow rN$ ▷ `Total budget`
$\mathbf{KV} \leftarrow \textsc{Cache}(\text{prompt}, \emptyset)$
$\mathbf{z}_1^{1:L} \sim q_{\text{prior}}$
for $t = 1, \frac{T-1}{T}, \ldots, \frac{1}{T}$ **do**
  $s \leftarrow t - \frac{1}{T}$
  $\mathcal{A}_{s|t} \leftarrow \{\ell \mid u^\ell(s) < u^\ell(t)\}$
  $\textsc{Logits}^{\mathcal{A}_{s|t}} \leftarrow \textsc{Denoiser}(\mathbf{z}_t^{\mathcal{A}_{s|t}}, u^{\mathcal{A}_{s|t}}(t), \mathbf{KV})$
  $\mathbf{z}_s^{\mathcal{A}_{s|t}} \leftarrow \textsc{Sampler}(\textsc{Logits}^{\mathcal{A}_{s|t}})$
  $\mathcal{C}_{s|t} \leftarrow \{\ell \in \mathcal{A}_{s|t} \mid u^\ell(s) = 0\}$
  $\mathbf{x}^{\mathcal{C}_{s|t}} \leftarrow \arg\max \textsc{Logits}^{\mathcal{C}_{s|t}}$
  $\mathbf{KV} \leftarrow \textsc{Cache}(\mathbf{x}^{\mathcal{C}_{s|t}}, \mathbf{KV})$
end for
return $\mathbf{x}^{1:L}$

3.3 Efficient Training

The diffusion loss term in (10) introduces a complexity for training: since the conditioning set varies by position, we cannot naively model a reverse diffusion step at each position with a single function evaluation. To account for this, Arriola et al. (2025a) propose a vectorized implementation where we train with sequences of length $2L$: the first part of the sequence corresponds to clean tokens $\mathbf{x}^{1:L}$ and the second part to noisy tokens $\mathbf{z}_t^{1:L}$. For each sample, we partition the noisy tokens into disjoint groups; each group represents a separate awake set. The attention mask $\mathcal{M} \in \{0,1\}^{2L \times 2L}$ can be broken into four $L \times L$ sub-masks: $\mathcal{M} = \begin{pmatrix} \mathcal{M}_{\text{CC}} & \mathbf{0} \\ \mathcal{M}_{\text{NC}} & \mathcal{M}_{\text{NN}} \end{pmatrix}$.

Clean-to-clean $\mathcal{M}_{\text{CC}}[\ell, \ell'] := \mathbf{1}[G_\varphi(\ell, \ell') \geq 0]$. A decoded token's KV entry is written once and reused for the remainder of generation, so it must be computable from what was available at the moment it was written; permitting $G_\varphi < 0$ would allow dependence on tokens that did not yet exist.

Noisy-to-clean For a reverse step from t to s, $\mathcal{M}_{\text{NC}}[\ell, \ell'] := \mathbf{1}[\ell' \in \mathcal{C}_t]$ for $\ell \in \mathcal{A}_{s|t}$. In our vectorized block and sliding window training layouts, each noisy group attends to the clean positions strictly preceding that group.

Noisy-to-noisy $\mathcal{M}_{\text{NN}}[\ell, \ell'] := 1$ if ℓ, ℓ' belong to the same denoising group, and 0 otherwise.

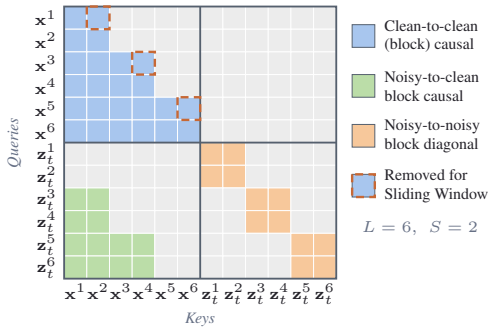


Figure 2: Clock Diffusion training masks. *Block*: clean-to-clean is block causal, a noisy query attends to clean keys strictly preceding its own block, and noisy queries attend bidirectionally within a block. *Sliding Window*: we remove the outlined clean-to-clean entries, making that quadrant causal.

3.4 Special Cases of Clock Diffusion

Full Sequence We first note that simply letting $\varphi(\ell) = 0$, for all ℓ, will yield $u^\ell(t) = t$ at every position, which recovers full sequence DLMs.

Block Clock Diffusion can also admit a continuous version of the block discrete diffusion method proposed in Arriola et al. (2025a). In Appendix B.5, we show how a specific choice of per-position phase defines local clocks, training, and sampling that correspond to block continuous diffusion.

Sliding Window Block diffusion is limited by the fact that we must wait for an entire block to complete denoising before updating the KV cache. To mitigate this, we propose Sliding Window Clock Diffusion, where we maintain an active window of at most S tokens that slides whenever the leftmost position transitions to $u^\ell(t) = 0$. We achieve this with the following per-position phase:

$$\varphi_{\text{SW}}(\ell) := \frac{L - \ell}{S} \implies u^\ell_{\text{SW}}(t) := \text{clip}_{[0,1]}\left(\frac{(L+S-1)t + \ell - L}{S}\right). \tag{11}$$

The phase gap now measures the distance between two positions in units of the window width: $G_\varphi(\ell, \ell') = \frac{L-\ell'}{S} - \frac{L-\ell}{S} = \frac{\ell - \ell'}{S}$. This translates to a straightforward modification of the block

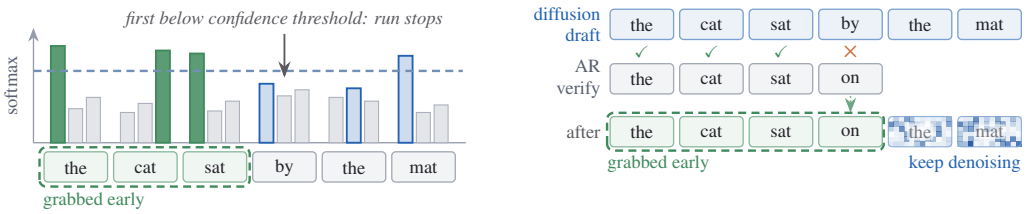


Figure 3: (*Left*) Cache Grab-Confidence. (*Right*) Cache Grab-SSD.

diffusion mask: $\mathcal{M}_{\text{CC}}$ reduces to pure causal attention. During training, we partition noisy positions into disjoint groups of S tokens, and each group attends to its own noisy positions and its preceding clean prefix (see Figure 2). In Appendix C.2, we show how to compute an unbiased estimate of the variational bound, which we derive in Appendix B.6. Training with causal attention on clean tokens allows us, at generation time, to commit and cache any leftmost prefix of an active window without waiting for the rest of the window to complete denoising, since we can always commit and cache these tokens with the causal AR mask. We leverage this fact in Section 3.6, below.

3.5 Clock Diffusion Language Models

Building on our framework, we combine *Sliding Window Clock Diffusion* with *Gaussian noise on token embeddings*, i.e., we use a learnable embedding matrix $E \in \mathbb{R}^{V \times d}$ as the embedder and perform diffusion over clean embeddings $\mathbf{e} := E^\top \mathbf{x}$, and we parameterize our denoising model as $\mathbf{e}_\theta := E^\top \mathbf{x}_\theta$, for $\mathbf{x}_\theta \in \Delta^V$. We also adopt several of the training techniques, such as learned noise schedules, that were used in a recent successful full sequence continuous DLM from Yang et al. (2026). Finally, we optimize a continuous-time formulation of the variational Sliding Window Clock Diffusion objective from (10). Together these elements form a new continuous DLM family that we call Clock Diffusion Language Models (**ClockDLMs**). As an ablation, below, we also train a block version, B-ClockDLMs. See Appendices B and C for full details.

3.6 Cache Grab Samplers: Accelerated Inference for ClockDLMs

In our sampling algorithm, any tokens whose local clocks transition to $u^\ell(s) = 0$ are committed and added to the cache (lines 9–11 of Algorithm 1). Here we introduce samplers that enlarge this set by optionally grabbing additional token positions for the cache. These **Cache Grab** samplers can thus accelerate ClockDLM inference by skipping function evaluations for early-committed tokens.

Cache Grab-Confidence In discrete diffusion, several works accelerate generation with little to no cost in quality by utilizing heuristics to select which token positions to generate. For example, Wu et al. (2025) propose to greedily unmask all positions for which the softmax output associated with the $\arg\max$ token exceeds some threshold. We can adapt this heuristic for ClockDLMs. Specifically, after every denoising step, we examine the probability associated with the $\arg\max$ token at every position, and we *early commit* any contiguous left-to-right positions that exceed our desired threshold, saving any remaining function evaluations in these tokens' denoising budgets. The accepted tokens can then be added to the KV cache and all remaining tokens from the active window shift left, continuing their denoising progress from where they left off, as new noisy tokens are added to the rightmost slots of the window. This sampler is depicted in Figure 3 and Algorithm 5 in Appendix E.1. In Appendix E.1, we present an adapted version of this sampler for B-ClockDLMs.

Cache Grab-Self Speculative Decoding Notably, our sliding window attention mask coincides with the one proposed in hybrid AR + Diffusion training (Gat et al., 2025; Fu et al., 2026). In particular, Fu et al. (2026) use this mask to enable a 'self speculative decoding' (SSD) mode, a variant of speculative decoding (Chen et al., 2023; Leviathan et al., 2023) where the diffusion drafter and AR verifier share all parameters. Inspired by this work, we similarly propose a Cache Grab-SSD sampler. As in Fu et al. (2026), we require a jointly trained AR prediction on the first half of the $2L$ training sequences, and at inference, we perform two function evaluations, one in diffusion mode to produce token drafts, and one in AR mode to verify and accept a prefix of the active window. Unlike the linear self-speculative sampler of Fu et al. (2026), which restarts drafting from all mask tokens, Cache Grab-SSD retains the noisy states and denoising progress of uncommitted positions. Figure 3 illustrates a single step of this sampler, and in Appendix E.2, we provide more details.

Table 1: Language modeling evaluation (perplexity; PPL ↓) of models trained on OWT. Diffusion values represent variational bounds ($\text{PPL}_{\leq}$). † Reported in Sahoo et al. (2024a). ‡ Reported in Sahoo et al. (2025). $ Reported in Arriola et al. (2025a). § Reported in Arriola et al. (2026). ¶ Reported in Zheng et al. (2026). * Reported in Yang et al. (2026). R Retrained from scratch.

		PPL (↓)
Autoregressive Baseline†		17.5
Discrete Diffusion Baselines		
UDLM‡ (Schiff et al., 2024)		27.4
Duo‡ (Sahoo et al., 2025)		25.2
SEDD Absorb† (Lou et al., 2024)		24.1
MDLM* (Sahoo et al., 2024a)		23.1
BD3LM (Arriola et al., 2025a)	$(S = 16)^{\$}$	22.3
	$(S = 4)^{\$}$	20.7
SW-SetDLM (Arriola et al., 2026)	$(S \leq 8)^{\S}$	22.4
Continuous Diffusion Baselines		
CADD¶ (Zheng et al., 2026)		23.1
LangFlow* (Chen et al., 2026)		32.2
Plaid* (Gulrajani et al., 2023)		24.4
RePlaidR (Yang et al., 2026)		21.3
Clock Diffusion (Ours)		
B-ClockDLM	$(S = 64)$	20.9
	$(S = 16)$	20.8
	$(S = 4)$	20.2
ClockDLM	$(S = 64)$	21.1
	$(S = 16)$	20.6
	$(S = 4)$	**19.5**

Table 2: Zero-shot, pass @ 1 accuracy (Acc %; ↑) on the GSM8K test set for models trained on the TinyGSM dataset. NFE values are means ± sample standard deviation. † Models and code from Deschenaux et al. (2026); $\mathbb{S}$-FLM accuracy quoted from that work. ‡ Models and code from Agarwal et al. (2026). R Retrained from scratch.

		NFEs/seq. (↓) Mean$_{\pm \text{SD}}$	Acc. % (↑) 0-shot, pass@1
*Autoregressive Baseline*R		$158.4_{\pm 48.7}$	64.3
Discrete Diffusion Baselines			
Duo† (Sahoo et al., 2025)		$1024_{\pm 0.0}$	19.0
w/softmax temp = 0.1†		$1024_{\pm 0.0}$	34.9
MDLMR (Sahoo et al., 2024a)		$112.4_{\pm 5.6}$	40.4
+ SAR decoding	$(S = 32)^{R}$	$184.1_{\pm 72.1}$	53.1
BD3LM (Arriola et al., 2025a)	$(S = 32)^{R}$	$177.7_{\pm 48.2}$	61.8
SW-SetDLM (Arriola et al., 2026)	$(S \leq 64)^{R}$	$174.1_{\pm 53.6}$	52.0
Continuous Diffusion Baselines			
CANDI† (Pynadath et al., 2025)		$1022.0_{\pm 2.8}$	0.4
FLM† (Lee et al., 2026)		$1024_{\pm 0.0}$	0.8
$\mathbb{S}$-FLM† (Deschenaux et al., 2026)		$1024_{\pm 0.0}$	18.0
FMLM+‡ (Agarwal et al., 2026)		$45.1_{\pm 33.3}$	21.0
MDLM Distill‡		$42.4_{\pm 37.0}$	26.1
MDLM Init‡		$39.2_{\pm 23.8}$	34.7
RePlaidR (Yang et al., 2026)		$1025.0_{\pm 0.0}$	44.5
Clock Diffusion (Ours)			
B-ClockDLM	$(S = 32)$	$190.1_{\pm 47.3}$	54.3
ClockDLM	$(S = 32)$	$188.1_{\pm 44.8}$	59.6
+ Cache Grab-Conf.		$59.5_{\pm 27.6}$	60.0
+ Cache Grab-SSD		$\mathbf{35.5}_{\pm 21.6}$	**67.6**

4 Experiments

Takeaway: ClockDLMs outperform previous continuous models while matching and exceeding the best SAR discrete models, achieving SoTA diffusion likelihood on standard benchmarks.

Baselines We compare to autoregressive (AR) and discrete diffusion models which include uniform diffusion (UDLM (Schiff et al., 2024), Duo (Sahoo et al., 2025)), masked diffusion (Sahoo et al., 2024a), and SAR masked diffusion models (Arriola et al., 2025a; 2026). We also compare against several notable continuous flow language models: LangFlow (Chen et al., 2026), FLM (Lee et al., 2026), $\mathbb{S}$-FLM (Deschenaux et al., 2026), and FMLM+ (Agarwal et al., 2026). The most direct comparisons to our work are Plaid (Gulrajani et al., 2023) / RePlaid (Yang et al., 2026) which are the full sequence continuous DLM analogs of our ClockDLMs.

4.1 Likelihood Evaluation

Experimental Setup Following previous works (Sahoo et al., 2024a; Arriola et al., 2025a), we train models on OpenWebText (Gokaslan et al., 2019) for 1M gradient update steps. We measure language modeling with perplexity (PPL ↓) and use variational lower bounds for diffusion models. In Appendix G.1, we also evaluate generative quality using MAUVE (Pillutla et al., 2021).

Results In Table 1, we demonstrate SoTA likelihoods compared to all diffusion models, including both continuous DLMs and full sequence and SAR discrete diffusion. Relative to BD3LM with $S = 4$, we reduce the PPL gap to the AR baseline by approximately 38%. Notably, both ClockDLM and B-ClockDLM outperform comparable SAR discrete diffusion.

4.2 Mathematical Reasoning

Experimental Setup We train relatively lightweight models on the TinyGSM dataset (Liu et al., 2023), which consists of math questions and Python code responses that execute a reasoning trace to produce the answer. We then evaluate on the GSM8K (Cobbe et al., 2021) test set.

Main Results In Table 2, we show that ClockDLM significantly outperforms all previous continuous DLMs, with up to ∼33% increase in accuracy, and we match the performance

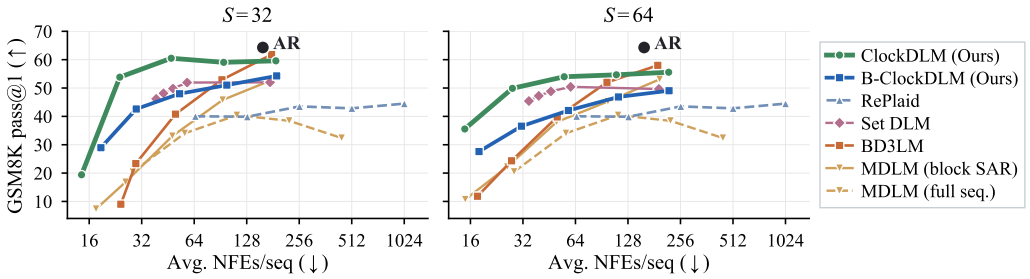


Figure 4: Accuracy–efficiency trade-offs on GSM8K. Zero-shot test pass@1 versus average NFEs/seq for models trained on TinyGSM. *(Left)* Active window size $S = 32$. *(Right)* $S = 64$.

of SAR discrete diffusion. When utilizing Cache Grab techniques we increase performance and throughput. Cache Grab-Confidence preserves accuracy with nearly 3× speedups; Cache Grab-SSD exceeds AR accuracy with the highest throughput among Table 2 configurations.

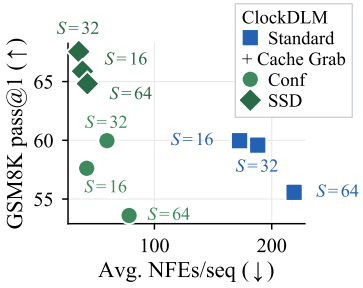


Figure 5: GSM8K Cache Grab. Cache Grab-Conf. achieves training-free speed-ups with minimal to no quality degradation. SSD unlocks improved performance & throughput.

Pareto Frontiers Figure 4 evaluates the performance-efficiency trade-off by sweeping over denoising budgets. ClockDLMs significantly push the frontier relative to the full sequence continuous diffusion baseline and match or exceed the performance of SAR discrete diffusion methods.

Cache Grab Improvements In Figure 5, we further explore the benefits of the Cache Grab samplers. Evaluating models trained at various active window sizes, $S \in \{16, 32, 64\}$, we show how Cache Grab-Conf. leads to minor to no accuracy drop off with ∼3-4× fewer NFEs/seq across window sizes and that Cache Grab-SSD unlocks improved performance with up to ∼5× fewer NFEs/seq. Notably, consistent with previous works that developed SSD for discrete diffusion, we also find that joint AR and diffusion loss training is complementary, increasing the performance of both AR and diffusion-only decoding (see Table 8 in Appendix G).

4.3 Summarization

Experimental Setup We train summarization models on CNN/DailyMail (Hermann et al., 2015; See et al., 2017), following Arriola et al. (2025b; 2026).

Results Table 3 reports NFEs/seq alongside ROUGE-1/2/L. ClockDLM ($S = 32$) improves all three ROUGE scores over RePlaid ($T = 1024$) with ∼ 8.5× fewer NFEs. In Figure 6, we again see the benefits of the accelerated Cache Grab-Confidence sampler. Reducing the confidence threshold from 0.99 to 0.7 requires ∼1.1-1.2× fewer NFEs/seq with at most a 1.3% reduction in ROUGE-1.

Table 3: Summarization metrics (ROUGE 1/2/L; ↑) on CNN/DailyMail. NFE values are means ± sample standard deviation. All models trained from scratch.

		NFEs/seq. (↓)	ROUGE (↑)		
		Mean$_{\pm SD}$	1	2	L
Autoregressive		$73.1_{\pm 13.4}$	35.9	14.8	25.0
Discrete Diffusion Baselines					
MDLM	(full seq.)	$180.0_{\pm 0.0}$	39.5	17.1	26.1
(Sahoo et al., 2024a)	($S = 64$)	$128.0_{\pm 0.0}$	40.1	18.1	27.2
	($S = 32$)	$96.0_{\pm 0.0}$	40.0	18.0	27.1
BD3LM	($S = 64$)	$126.9_{\pm 19.4}$	39.8	17.7	27.2
(Arriola et al., 2025a)	($S = 32$)	$110.7_{\pm 9.3}$	39.4	17.5	27.0
SW-SetDLM	($S \leq 128$)	$74.9_{\pm 14.2}$	38.8	17.1	27.1
(Arriola et al., 2026)	($S \leq 64$)	$72.2_{\pm 16.2}$	39.7	17.9	27.8
Continuous Diffusion Baseline					
RePlaid	($T = 1024$)	$1025.0_{\pm 0.0}$	32.2	12.7	22.1
(Yang et al., 2026)	($T = 128$)	$129_{\pm 0.0}$	30.4	11.1	20.7
Clock Diffusion (Ours)					
B-ClockDLM	($S = 64$)	$120.2_{\pm 32.0}$	34.3	12.7	22.9
	($S = 32$)	$91.3_{\pm 18.3}$	33.6	11.3	21.8
ClockDLM	($S = 64$)	$152.4_{\pm 0.6}$	34.0	11.7	21.7
	($S = 32$)	$120.3_{\pm 0.8}$	36.0	14.1	23.8

4.4 Ablations

Block vs. Sliding Window We often find that ClockDLMs, which are based on sliding window clocks, outperform their block counterparts B-ClockDLMs. This may reflect the additional flexibil-

ity of ClockDLMs, which have a more left-to-right decoding scheme even within active windows compared to block diffusion.

Effect of Window Size We find the expected pattern that in almost all cases moving to smaller blocks / window sizes improves performance as generation becomes more like AR. We note that our results are robust to window size, e.g., at matched blocks for BD3LM, we consistently improve language modeling perplexity (Table 1) and the Pareto frontier of quality vs. efficiency (Figure 4).

Varying Confidence Threshold In Figure 6, we see that speed gains grow as the confidence threshold is lowered, as expected, and we see a fairly monotonic degradation in quality. In Table 7 in Appendix G.2, this monotonic relationship is less pronounced.

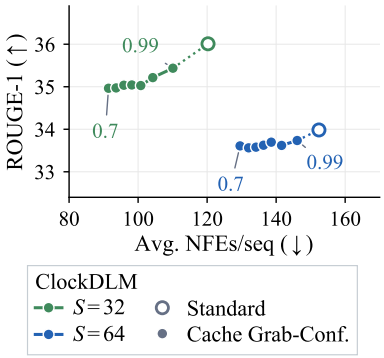


Figure 6: Cache Grab-Confidence on CNN/DailyMail. ROUGE-1 vs. NFEs/seq for ClockDLMs, sweeping the early-commit confidence threshold. Hollow markers use no early commit, i.e., rows in Table 3.

5 Related Work

Our work is most related to the recent continuous DLMs that revived token embedding diffusion with modernized architectures and training recipes (Chen et al., 2026; Yang et al., 2026). We also leverage vectorized training methods introduced in block discrete diffusion (Arriola et al., 2025a). Han et al. (2022) used SAR decoding with diffusion over logits, but lagged in quality relative to discrete diffusion and did not propose our efficient training and KV cache sampling.

Position-dependent Noise Schedules Previous diffusion works have explored position-dependent noise. AR Diffusion (Wu et al., 2023) biased noise across the full sequence to have leftmost tokens be less noisy, and Diffusion Forcing (Chen et al., 2024) generalized this method. Most closely related is Rolling Diffusion (Ruhe et al., 2024), which formalized a process akin to our Sliding Window Clock Diffusion. However, unlike these works, we propose efficient vectorized training and KV cache sampling. Furthermore, we complement our sampling algorithm with the Cache Grab samplers, which significantly boost throughput. Finally, both Chen et al. (2024) and Ruhe et al. (2024) examined continuous data, such as video frames, while we have adapted this method to language and derived each component of the variational bound in this setting.

6 Discussion & Conclusion

In this work, we introduced a semi-autoregressive formulation for continuous diffusion language modeling that relies on position-dependent noise schedules that determine which tokens are actively being denoised at any given diffusion step. We complement this parameterization with efficient training recipes and KV cache sampling and call this framework Clock Diffusion. We demonstrated two instantiations of Clock Diffusion that enable block and sliding window SAR generation. Additionally, we leveraged the sliding window formulation to develop accelerated sampling techniques that adapt confidence-threshold and self-speculative decoding from discrete diffusion. We introduced models that use Clock Diffusion with Gaussian noise on token embeddings, ClockDLMs, and trained them on unconditional and conditional language modeling tasks. We attained SoTA language modeling performance measured in both perplexity and downstream benchmark metrics. We consistently outperformed full sequence continuous language models on both performance and efficiency, demonstrating the usefulness of our SAR parameterization and KV cache enabled sampling.

Limitations The improved language modeling performance of our method comes at the cost of decoding flexibility, which may be beneficial in certain domains that lack a more strict left-to-right bias. Additionally, we have presently only explored relatively smaller scale models.

Future Directions In future efforts, we intend to address the limitations of our work with extensions to any-order sliding window implementations. We also plan to test our method at larger scales and explore the efficacy of leveraging pre-trained AR and discrete diffusion models. Finally, we intend to experiment with hybrid discrete and continuous parameterizations.

Reproducibility Statement

We provide the full derivation for the variational objective employed by our method in Appendix A and the models we trained are explicitly defined in Appendix B. All relevant training details and training and evaluation hyperparameter settings are explicitly laid out in Appendices C and F.

AI Use Statement

In this work, we used generative AI tools, specifically large language models (LLMs), to assist in the derivation of the mathematical formulation presented in Section 3 and Appendix A. Generative AI was used in developing code to run experiments and in producing the tables and figures that report our results. We also used LLMs to summarize existing literature, edit the formatting of this work, and format references. We further used LLMs to assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, and interpret experimental results. We have not used generative AI tools to generate synthetic data sets, assist with translation, or clean and reformat datasets. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

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.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. Smollm - blazingly fast and remarkably powerful, 2024.

Marianne Arriola and Volodymyr Kuleshov. Set diffusion: Interpolating token orderings between autoregression and diffusion for fast and flexible decoding. *arXiv preprint arXiv:2607.01775*, 2026.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025a.

Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. *Advances in Neural Information Processing Systems*, 38:15612–15644, 2025b.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 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.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

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.

Oscar Davis, Anastasiia Filippova, Pierre Ablin, Victor Turrisi, Amitis Shidani, Marco Cuturi, and Louis Béthune. Scaling categorical flow maps. *arXiv preprint arXiv:2605.07820*, 2026.

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.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator. *arXiv preprint arXiv:2509.04185*, 2025.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*, 2022.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701, 2015. URL http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

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.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pages 19274–19286. PMLR, 2023.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343, 2022.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

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.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning*, 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. *arXiv preprint arXiv:2502.09992*, 2025.

The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

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

Krishna Pillutla. mauve. https://github.com/krishnap25/mauve, 2021. Accessed: 2025-05-14.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34:4816–4828, 2021.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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.

David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogeboom. Rolling diffusion models. *arXiv preprint arXiv:2402.09470*, 2024.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Mariano Marroquin, Justin T Chiu, Alexander M Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=L4uaAR4ArM.

Subham Sekhar Sahoo, Aaron Gokaslan, Christopher De Sa, and Volodymyr Kuleshov. Diffusion models with learned adaptive noise. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=loMa99A4p8.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The diffusion duality. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2025.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-torre, Bernardo P de Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. *arXiv preprint arXiv:2412.10193*, 2024.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://www.aclweb.org/anthology/P17-1099.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 2526–2547, 2025.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

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

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

Zhihan Yang, Wei Guo, Shuibai Zhang, Subham Sekhar Sahoo, Yongxin Chen, Arash Vahdat, Morteza Mardani, and John Thickstun. Continuous diffusion scales competitively with discrete diffusion for language. *arXiv preprint arXiv:2605.18530*, 2026.

Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. Continuously augmented discrete diffusion model for categorical generative modeling. In *International Conference on Learning Representations*, volume 2026, pages 34212–34242, 2026.

CONTENTS

1 Introduction **2**

2 Background **2**

3 Clock Diffusion **3**

3.1 Parameterization 3

3.2 Key-Value Cache Sampling 4

3.3 Efficient Training 5

3.4 Special Cases of Clock Diffusion 5

3.5 Clock Diffusion Language Models 6

3.6 Cache Grab Samplers: Accelerated Inference for ClockDLMs 6

4 Experiments **7**

4.1 Likelihood Evaluation 7

4.2 Mathematical Reasoning 7

4.3 Summarization 8

4.4 Ablations 8

5 Related Work **9**

6 Discussion & Conclusion **9**

A Clock Diffusion Variational Bound **17**

B Clock Diffusion Language Models **18**

B.1 Clock Diffusion with Gaussian Noise 18

B.2 Learned Embedding Parameterizations 18

B.3 Variational Clock Diffusion: Continuous-time Limit 19

B.4 Inherited RePlaid Components 19

B.5 Block Clock Diffusion Language Models 21

B.6 Sliding Window Clock Diffusion Language Models 22

C Additional Training & Evaluation Details for Clock Diffusion Language Models **23**

C.1 Block Training 24

C.2 Sliding Window Training & Evaluation 24

D Additional Clock Diffusion Sampling Details **25**

E Cache Grab Sampler Algorithms **26**

E.1 Cache Grab-Confidence 26

E.2 Cache Grab-SSD 27

F Additional Experimental Setup Details **28**

F.1 OpenWebText 28

F.2 Math Reasoning Benchmark 29

F.3 Summarization 30

G Additional Experimental Results **31**

G.1 Open Ended Generation 31

G.2 Cache Grab-Confidence for ClockDLMs & B-ClockDLMs on GSM8K 32

G.3 Effect of SSD Training 33

G.4 Generated Sample Examples 34

H Assets **39**

A Clock Diffusion Variational Bound

In this section, we derive the Clock Diffusion variational bound. We reuse elements from the derivation in Ruhe et al. (2024), which demonstrates how splitting each latent sequence $\mathbf{z}_t^{1:L}$ into active (*awake*) and inactive (*snoozed*) positions simplifies the diffusion loss component of the variational objective.

In the finite-time setting, we have $T \in \mathbb{N}$ latent sequences $\mathbf{z}_{0:1}^{1:L} := (\mathbf{z}_{t(i)}^{1:L})_{i=0}^{T-1}$, where we let $t : \{0, \dots, T-1\} \to [0, 1]$ and $0 = t(0) < t(1) < \dots < t(T-1) = 1$, and $s(i) := t(i-1)$, for $i \in \{1, \dots, T-1\}$.

We start by explicitly stating our assumed true reverse process:

$$q(\mathbf{z}_{0:1}^{1:L} \mid \mathbf{x}^{1:L}) := q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L}) \prod_{i=1}^{T-1} q(\mathbf{z}_{s(i)}^{1:L} \mid \mathbf{z}_{t(i)}^{1:L}, \mathbf{x}^{1:L}). \tag{12}$$

To simplify the notation, in what follows, we drop the dependence of s, t on i.

Our aim is to optimize the maximum likelihood, or equivalently, the minimum negative log likelihood, objective. We apply the standard importance weighting and Jensen's inequality techniques to this objective to derive the diffusion variational bound (Sohl-Dickstein et al., 2015):

$$\begin{aligned}
&\mathbb{E}_{q_{\text{data}}(\mathbf{x}^{1:L})}\left[-\log p_\theta(\mathbf{x}^{1:L})\right] = \mathbb{E}_{q_{\text{data}}(\mathbf{x}^{1:L})}\left[-\log \int_{\mathbf{z}_{0:1}^{1:L}} p_\theta(\mathbf{x}^{1:L}, \mathbf{z}_{0:1}^{1:L}) d\mathbf{z}_{0:1}^{1:L}\right] \\
&= \mathbb{E}_{q_{\text{data}}(\mathbf{x}^{1:L})}\left[-\log \int_{\mathbf{z}_{0:1}^{1:L}} \frac{p_\theta(\mathbf{x}^{1:L}, \mathbf{z}_{0:1}^{1:L})}{q(\mathbf{z}_{0:1}^{1:L} \mid \mathbf{x}^{1:L})} q(\mathbf{z}_{0:1}^{1:L} \mid \mathbf{x}^{1:L}) d\mathbf{z}_{0:1}^{1:L}\right] \\
&\leq \mathbb{E}_{q_{\text{data}}(\mathbf{x}^{1:L})}\left[\log q(\mathbf{z}_{0:1}^{1:L} \mid \mathbf{x}^{1:L}) - \log p_\theta(\mathbf{x}^{1:L}, \mathbf{z}_{0:1}^{1:L})\right] \\
&= \mathbb{E}_{q_{\text{data}}(\mathbf{x}^{1:L})}\left[\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{recons}} + \mathcal{L}_{\text{diffusion}}^T\right].
\end{aligned} \tag{13}$$

Using the factorizations from (12) and those from Section 3.1, the prior loss $\mathcal{L}_{\text{prior}}$, reconstruction loss $\mathcal{L}_{\text{recons}}$, and finite-time diffusion loss $\mathcal{L}_{\text{diffusion}}^T$ are defined as follows:

$$\mathcal{L}_{\text{prior}} := \text{KL}[q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L}) || p_\theta(\mathbf{z}_1^{1:L})] \tag{14}$$

$$\mathcal{L}_{\text{recons}}^T := \sum_{i=1}^{T-1} \mathbb{E}_{q(\mathbf{z}_s^{\mathcal{A}_{s|t}} | \mathbf{x}^{1:L})} \sum_{\ell \in \mathcal{C}_{s|t}} -\log p_\theta(\mathbf{x}^\ell \mid \mathbf{z}_s^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t}) \tag{15}$$

$$\mathcal{L}_{\text{diffusion}}^T := \sum_{i=1}^{T-1} \mathbb{E}_{q(\mathbf{z}_t^{1:L} | \mathbf{x}^{1:L})} \text{KL}\left[q(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L}) || p_\theta(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t})\right]. \tag{16}$$

Simplifying the Diffusion Loss We first note that we parameterize the awake transition as a product over positions, with each factor conditioned on all awake latents and previously revealed tokens:

$$p_\theta(\mathbf{z}_s^{\mathcal{A}_{s|t}} \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t}) := \prod_{\ell \in \mathcal{A}_{s|t}} p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t}).$$

Together with the deterministic transitions at snoozed positions, this factorization lets us simplify each term in (16):

$$\begin{aligned}
&\text{KL}\left[q(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L}) || p_\theta(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t})\right] = \mathbb{E}_{q(\mathbf{z}_s^{1:L} | \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L})}\left[-\log \frac{p_\theta(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{\mathcal{C}_t})}{q(\mathbf{z}_s^{1:L} \mid \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}_s^{1:L} | \mathbf{z}_t^{1:L}, \mathbf{x}^{1:L})}\left[\sum_{\ell \in \mathcal{A}_{s|t}} -\log \frac{p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})}{q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{1:L})} - \underbrace{\sum_{\ell \in \mathcal{Z}_{s|t}} \log \frac{p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell)}{q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell)}}_{=0}\right] \\
&= \sum_{\ell \in \mathcal{A}_{s|t}} \mathbb{E}_{q(\mathbf{z}_s^\ell | \mathbf{z}_t^\ell, \mathbf{x}^\ell)}\left[-\log \frac{p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})}{q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell)}\right].
\end{aligned} \tag{17}$$

Plugging (17) into (16), we get:

$$\mathcal{L}_{\text{diffusion}}^{T,\text{Clock}} = \sum_{i=1}^{T-1} \mathbb{E}_{q(\mathbf{z}_t^{1:L}|\mathbf{x}^{1:L})} \sum_{\ell \in \mathcal{A}_{s|t}} \text{KL}[q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) || p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})]. \tag{18}$$

B Clock Diffusion Language Models

In this section we build on our framework to create a family of models that use Gaussian noise applied to token embeddings and leverage the parameterization and training recipes from RePlaid (Yang et al., 2026) in conjunction with Clock Diffusion. When using the sliding window formulation, we call this family of models Clock Diffusion Language Models (ClockDLMs). We also define block versions that we denote as B-ClockDLMs.

B.1 Clock Diffusion with Gaussian Noise

When adding Gaussian noise, we have the following functional form of the forward corruption process and true posterior (Sohl-Dickstein et al., 2015):

$$q(\mathbf{z}_t^\ell \mid \mathbf{x}^\ell) = \mathcal{N}(\alpha_t \mathbf{x}^\ell, \sigma_t^2 \mathbf{I}) \quad \text{and} \quad q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) = \mathcal{N}\left(\frac{\alpha_{t|s}\sigma_s^2}{\sigma_t^2}\mathbf{z}_t^\ell + \frac{\alpha_s \sigma_{t|s}^2}{\sigma_t^2}\mathbf{x}^\ell, \frac{\sigma_{t|s}^2 \sigma_s^2}{\sigma_t^2}\mathbf{I}\right), \tag{19}$$

where $\alpha_{t|s} := \alpha_t/\alpha_s, \sigma_{t|s}^2 := \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2$, and $\alpha_t := \alpha(t)$ and $\sigma_t := \sigma(t)$ are monotonically decreasing and increasing functions in t, respectively, that dictate the amount of signal and noise, respectively, present in a given latent variable. Thus we have $\mathbf{z}_0^{1:L}$ as the least noisy latent sequence and $\mathbf{z}_1^{1:L}$ as the most noisy. An important quantity of interest is the signal-to-noise ratio, which we define as $\text{SNR}(t) := \alpha_t^2/\sigma_t^2$.

In Clock Diffusion, we adjust all of the noise schedule quantities to be position dependent, that is: $\alpha_t^\ell := \alpha(u^\ell(t)), \sigma_t^\ell := \sigma(u^\ell(t))$, and $\text{SNR}^\ell(t) := \text{SNR}(u^\ell(t)) = \alpha(u^\ell(t))^2/\sigma(u^\ell(t))^2$.

With this, each KL term in (18) becomes (Kingma et al., 2021):

$$\text{KL}[q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) || p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})] = \tfrac{1}{2}(\text{SNR}^\ell(s) - \text{SNR}^\ell(t)) \, \|\mathbf{x}^\ell - \mathbf{x}_\theta^\ell(\mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})\|_2^2. \tag{20}$$

Finally, we take as our prior $p_\theta(\mathbf{z}_1^{1:L}) := \prod_{\ell=1}^L \mathcal{N}(\mathbf{0}, \mathbf{I})$. Using $u^\ell(1) = 1$, for all ℓ, the prior loss, as a KL between two Gaussian distributions, has a closed form solution:

$$\mathcal{L}_{\text{prior}} = \frac{1}{2} \sum_{\ell=1}^{L} \left[\alpha(1)^2 \|\mathbf{x}^\ell\|_2^2 + d\left(\sigma(1)^2 - 1 - \log \sigma(1)^2\right)\right]. \tag{21}$$

B.2 Learned Embedding Parameterizations

In our experiments, we choose to instantiate continuous diffusion for discrete data using Gaussian noise applied to token embeddings, as in Li et al. (2022); Dieleman et al. (2022); Gulrajani et al. (2023); Lee et al. (2026); Yang et al. (2026). In particular, let $E \in \mathbb{R}^{V \times d}$ denote an embedding matrix that maps one-hot tokens and probability vectors in the simplex to d-dimensional token embeddings. As described in Section 2, we let $\mathbf{x}^\ell \in \{0,1\}^V \cap \Delta^V$ denote a one-hot vector and $\mathbf{e}^\ell := E^\top \mathbf{x}^\ell \in \mathbb{R}^d$ denote its token embedding. We use the notation $\mathbf{e}^{1:L} := E[\mathbf{x}^{1:L}] := (E^\top \mathbf{x}^1, \ldots, E^\top \mathbf{x}^L) \in \mathbb{R}^{L \times d}$ for a sequence of L embeddings. Following Gulrajani et al. (2023); Yang et al. (2026), we let $\mathbf{x}_\theta^\ell \in \Delta^V$ be the softmax output of a neural network that predicts logits over the vocabulary at position ℓ and our parameterized prediction of the clean token embeddings is $\mathbf{e}_\theta^{1:L} := E[\mathbf{x}_\theta^{1:L}]$, with $\mathbf{e}_\theta^\ell := E^\top \mathbf{x}_\theta^\ell$, for all ℓ. We normalize each row of E to unit L^2 norm. Since $\mathbf{x}^\ell$ is one-hot, $\|\mathbf{e}^\ell\|_2 = 1$. The predicted embedding $\mathbf{e}_\theta^\ell$ is a convex combination of normalized embedding rows, so $\|\mathbf{e}_\theta^\ell\|_2 \leq 1$.

Our equations from Appendix B.1 are updated as follows:

$$q(\mathbf{z}_t^\ell \mid \mathbf{x}^\ell) = \mathcal{N}(\alpha_t \mathbf{e}^\ell, \sigma_t^2 \mathbf{I}), \qquad q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) = \mathcal{N}\left(\tfrac{\alpha_{t|s}\sigma_s^2}{\sigma_t^2}\mathbf{z}_t^\ell + \tfrac{\alpha_s \sigma_{t|s}^2}{\sigma_t^2}\mathbf{e}^\ell, \tfrac{\sigma_{t|s}^2 \sigma_s^2}{\sigma_t^2}\mathbf{I}\right), \tag{22}$$

$$\mathrm{KL}[q(\mathbf{z}_s^\ell \mid \mathbf{z}_t^\ell, \mathbf{x}^\ell) || p_\theta(\mathbf{z}_s^\ell \mid \mathbf{z}_t^{\mathcal{A}_{s|t}}, \mathbf{x}^{\mathcal{C}_t})] = \tfrac{1}{2}(\mathrm{SNR}^\ell(s) - \mathrm{SNR}^\ell(t)) \, \|\mathbf{e}^\ell - \mathbf{e}_\theta^\ell\|_2^2, \tag{23}$$

$$\text{and} \qquad \mathcal{L}_{\text{prior}} = \frac{1}{2}\sum_{\ell=1}^{L}\left[\alpha(1)^2 + d\left(\sigma(1)^2 - 1 - \log \sigma(1)^2\right)\right], \tag{24}$$

Note that $\mathbf{z}_s^\ell, \mathbf{z}_t^\ell \in \mathbb{R}^d$ and that $\mathbf{I}$ is a $d \times d$ identity matrix.

Finally, the total optimizable parameters consist of both neural network parameters θ and the learned embedding matrix E.

B.3 Variational Clock Diffusion: Continuous-time Limit

As in previous works (Kingma et al., 2021; Sahoo et al., 2024a;b), we can explore the variational lower bound in the continuous time limit of the diffusion process, $T \to \infty$. We now examine each of the loss terms from (13) in this limit.

Prior Loss First, note that $\mathcal{L}_{\text{prior}}$ is unaffected by this limit, since it only depends on the noise schedule values at the noisy endpoint $\alpha(1)$ and $\sigma(1)$.

Reconstruction Loss For the reconstruction loss, we show that there exists a finite grid on which the reconstruction loss is invariant under further refinement.

Define each position's completion time as $\tau_\ell := \max\{t : u^\ell(t) = 0\}$, i.e., the time at which position ℓ completes denoising as global time decreases from 1 to 0. Now, for any time $t \in [0, 1)$ define:

$$\mathcal{A}_{t^+} := \lim_{\varepsilon \downarrow 0} \mathcal{A}_{t \,|\, t+\varepsilon} \qquad \text{and} \quad \mathcal{C}_{t^+} := \lim_{\varepsilon \downarrow 0} \mathcal{C}_{t+\varepsilon}. \tag{25}$$

These limits exist because these sets are nested as ε decreases and contain only finitely many positions, i.e., they are subsets of $\{1, \ldots, L\}$; consequently, they are constant for all sufficiently small positive ε.

Suppose we have a grid of T latents such that for all $\ell \in \{1, \ldots, L\}$, $t(i) = \tau_\ell$, for some $i \in \{0, \ldots, T-1\}$. Then for any refinements of this finite grid containing every completion time, with the maximum interval length tending to zero, the reconstruction loss will converge to a stable value that is equal to the following:

$$\mathcal{L}_{\text{recons}}^{\infty} = \sum_{\ell=1}^{L} \mathbb{E}_{q(\mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell^+}} \mid \mathbf{x}^{1:L})}\left[-\log p_\theta\left(\mathbf{x}^\ell \,\middle|\, \mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell^+}}, \mathbf{x}^{\mathcal{C}_{\tau_\ell^+}}\right)\right]. \tag{26}$$

Intuitively this loss is simply a sum of the per-token cross-entropy loss for our distribution over the token $\mathbf{x}^\ell$ given any active tokens and any clean token that previously completed denoising at the time when position ℓ finishes denoising.

Diffusion Loss For the diffusion loss, our derivation follows directly from Kingma et al. (2021) and Ruhe et al. (2024), and we have:

$$\mathcal{L}_{\text{diffusion}}^{\infty,\text{Clock}} := \int_{t=0}^{t=1}\left[\mathbb{E}_{q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L})} \sum_{\ell \in \mathcal{A}_{t^+}} -\tfrac{1}{2}\mathrm{SNR}'(u^\ell(t)) \, \dot{u}^\ell(t) \, \|\mathbf{e}^\ell - \mathbf{e}_\theta^\ell(\mathbf{z}_t^{\mathcal{A}_{t^+}}, \mathbf{x}^{\mathcal{C}_t})\|_2^2\right] dt, \tag{27}$$

where SNR' and $\dot{u}^\ell$ denote the time derivatives of SNR and u^ℓ, respectively. We assume that the local clocks are absolutely continuous and the base SNR schedule is continuously differentiable for these quantities to exist, almost surely for $t \in [0, 1]$.

B.4 Inherited RePlaid Components

In addition to using Gaussian noise on token embeddings, ClockDLMs leverage several of the training techniques from RePlaid (Yang et al., 2026), a recent promising continuous DLM.

Variance-Preserving Latents and γ Conditioning We adopt the variance-preserving formulation, i.e., $\alpha_t^2 + \sigma_t^2 = 1$, and we use the negative log-SNR parameterization of the noise schedule, $\gamma_t := \gamma(t) := -\log \mathrm{SNR}(t)$, so that

$$\alpha_t = \sqrt{\mathrm{sigmoid}(-\gamma_t)}, \qquad \sigma_t = \sqrt{1-\alpha_t^2} = \sqrt{\mathrm{sigmoid}(\gamma_t)}. \tag{28}$$

We provide γ_t as the noise conditioning variable to the denoising network. Unlike RePlaid, γ_t varies per position for our models, so we use $\gamma_t^{1:L} := \big(\gamma(t^\ell)\big)_{\ell=1}^{L}$ to denote per-position noise conditioning.

Learned Noise Schedule As in Yang et al. (2026), γ_t is a learnable monotone function of t that is trained jointly with the model to reduce the variance of the variational objective, and we use their efficient variance-reduction estimator (see Figure 8 in Yang et al. (2026)): the gradient reaching the schedule's shape parameters is rescaled so that the schedule minimizes the second moment, rather than the mean, of the diffusion loss. We consider two monotone parameterizations of γ_t. The first is the monotone network of Kingma et al. (2021) used by RePlaid, with positively constrained weights. The second, which we found more stable on our conditional tasks, controls the schedule through its slope. Specifically, fix K equally spaced times spanning $[0, 1]$ and learn a value of $\log m(t)$ at each one. We linearly interpolate these values between neighboring times and exponentiate the result to obtain a positive function $m(t)$. We then define

$$\gamma_t = \gamma_0 + (\gamma_1 - \gamma_0)\frac{\int_0^t m(u)\,du}{\int_0^1 m(u)\,du}. \tag{29}$$

Endpoints γ_0, γ_1 can also be learned jointly with the shape. For some settings we fix these endpoints and for some we allow them to be learned, as in RePlaid. These schedule settings are reported in Appendix F.

Embedding Bottleneck Following RePlaid, we represent each token by a learned embedding of dimension $d = 16$ and apply Gaussian diffusion in this embedding space. We normalize each row of the embedding matrix E to unit L^2 norm. Before passing a corrupted embedding $\mathbf{z}_t^\ell$ to the Transformer, we divide it by $\sqrt{(\alpha_t^\ell)^2/d + (\sigma_t^\ell)^2}$. This keeps its expected squared norm equal to d across noise levels. We then apply a learned linear projection without bias to map the d-dimensional latent to the Transformer's hidden dimension. Clean context embeddings are multiplied by $\sqrt{d}$ and passed through the same projection.

We use separate optimizer parameter groups for E and the input projections, as specified in Appendix F.

Self-Conditioning We employ self-conditioning (Chen et al., 2022), where during inference, the denoising model is also conditioned on the output from the previous denoising forward model evaluation. During training, we apply self-conditioning to a deterministically selected quarter of the examples in each global batch. For these examples, we first run the denoiser without self-conditioning under `no_grad` and compute its posterior-mean clean embedding $\mathbf{e}_\theta^{1:L} = E[\mathbf{x}_\theta^{1:L}]$, without renormalizing its length. We then use this estimate to condition a second forward pass on the same corrupted input, diffusion times, and attention mask, and compute the training loss from that second pass. For these training examples, we stop gradients through the input embeddings (both corrupted token embeddings $\mathbf{z}_t^{1:L}$ and clean context embeddings) and the schedule quantities. The second pass still trains the denoiser and its input projections.

The self-conditioning input is zero on the clean half of the $2L$ canvas and on examples not selected for self-conditioning. On batch samples that receive self-conditioning, we supply the estimate only to the noisy half. For conditional tasks, any fixed conditioning tokens present in that half use their known clean embeddings instead of model predictions.

Validation uses the same two-pass procedure for every example.

At inference, generated positions start with zero self-conditioning and subsequent denoising steps reuse the preceding clean embedding estimate without an additional bootstrap pass.

We implement self-conditioning as in RePlaid by multiplying the self-conditioning embedding by $\sqrt{d}$, applying a separate bias-free linear projection to the Transformer width, and adding it to the projected input. This projection is initialized to zero.

Batch Splitting During training, evaluating both diffusion and reconstruction loss terms on every example would require two backbone forwards per example, plus any self-conditioning bootstrap forwards. Instead, as in RePlaid (see Algorithm 1 in Yang et al. (2026)), during training, we split each global batch into disjoint subsets, evaluating the diffusion term on one subset and the reconstruction term on the other. We allocate the split as follows: with n examples in the global batch, we assign

$$n_{\text{recons}} = \text{round}\left(n \cdot \frac{\hat{\sigma}_{\text{recons}}}{\hat{\sigma}_{\text{recons}} + \hat{\sigma}_{\text{diffusion}}} \right) \tag{30}$$

examples to the reconstruction term, clipped to $[1, n-1]$, where $\hat{\sigma}_{\text{recons}}$ and $\hat{\sigma}_{\text{diffusion}}$ are standard deviations estimated from de-biased exponential moving averages (decay 0.997) of the first and second loss moments of reconstruction and diffusion loss terms, respectively. Each subset's contribution is rescaled by the inverse of its sampling fraction so that the resulting estimate of the full objective remains unbiased.

Output Logit Prior We also use the 'output logit prior' parameterization proposed in Gulrajani et al. (2023) and adpoted in Yang et al. (2026): rather than applying softmax directly to the neural network's logits, we add to these logits the analytic Gaussian log-likelihood of the current latent under each vocabulary embedding. Concretely, let $h_\theta^\ell \in \mathbb{R}^V$ denote the raw logits emitted by the denoiser at position ℓ. We form the prior logits

$$\pi_t^\ell[v] := \log \mathcal{N}\big(\mathbf{z}_t^\ell;\, \alpha_t^\ell E_{v,:},\, (\sigma_t^\ell)^2 \mathbf{I}\big) \;\overset{c}{=}\; \frac{\alpha_t^\ell}{(\sigma_t^\ell)^2} \langle \mathbf{z}_t^\ell,\, E_{v,:} \rangle - \frac{(\alpha_t^\ell)^2}{2(\sigma_t^\ell)^2} \big\| E_{v,:} \big\|_2^2, \qquad v \in \{1, \dots, V\}, \tag{31}$$

where $\overset{c}{=}$ denotes equality up to an additive term that is constant in v and therefore absorbed by the softmax. Because we L^2-normalize the embedding rows, the second term in (31) is constant across the vocabulary and can be dropped, so the prior reduces to the scaled inner product $\pi_t^\ell = \frac{\alpha_t^\ell}{(\sigma_t^\ell)^2} E\mathbf{z}_t^\ell$.

Soft-capping One notable departure from RePlaid's formulation is that we apply soft-capping (Team et al., 2024) to our logits. Thus, combined with the output logit prior from the paragraph above, our final softmax output takes the following form:

$$\mathbf{x}_\theta^\ell := \text{softmax}\big(\kappa \tanh\big(h_\theta^\ell / \kappa\big) + \pi_t^\ell\big), \qquad \kappa = 30. \tag{32}$$

B.5 Block Clock Diffusion Language Models

Here we present a block instantiation of Clock Diffusion. As noted in Section 3.4, we need to specify the per-position phase and then local clocks, training mask, and sampling follow accordingly. Specifically, for a sequence of L tokens, we can split the sequence into B non-overlapping blocks of $S := L/B$ tokens. Letting $b(\ell) := 1 + \lfloor (\ell - 1)/S \rfloor$ be the block index of position ℓ, we have:

$$\varphi_{\text{B}}(\ell) := B - b(\ell) \implies u_{\text{B}}^\ell(t) := \text{clip}_{[0,1]}\left(B(t-1) + b(\ell)\right), \tag{33}$$

$\varphi_{\text{B}}(\ell)$ can be interpreted as the number of blocks that come after ℓ's own block. Under this formulation, the phase gap for two tokens simply becomes their difference in block index:

$$G_\varphi(\ell, \ell') = (B - b(\ell')) - (B - b(\ell)) = b(\ell) - b(\ell') \begin{cases} \geq 1 & \ell' \text{ decodes earlier than } \ell, \\ = 0 & \ell, \ell' \text{ are in the same block}, \\ \leq -1 & \ell' \text{ decodes later than } \ell. \end{cases}$$

Consequently, our training mask becomes the one depicted in Figure 2. For sampling, $r = B$, so Algorithm 1 runs for $T = BN$ model calls, with one block awake at any time. Every N steps, an entire block completes in lockstep and is added to the KV cache. With this formulation, vectorized training and sampling coincide with the methods of Arriola et al. (2025a).

Now, using the block clocks defined in (33), we show what each loss term becomes for the block instantiation of ClockDLMs. First, for a sequence $\mathbf{x}^{1:L}$, for each $b \in \{1, \dots, B\}$, let $\mathbf{x}^b := \mathbf{x}^{(b-1)S+1:bS}$ and $\mathbf{x}^{<b} := \mathbf{x}^{1:(b-1)S}$ denote block b and its preceding clean context, respectively.

Under the block clocks, for $t \in \left(\frac{B-b}{B}, \frac{B-b+1}{B}\right)$, the active and completed sets are:

$$\mathcal{A}_{t^+} = \{(b-1)S+1, \ldots, bS\}, \qquad \mathcal{C}_t = \{1, \ldots, (b-1)S\}. \tag{34}$$

All positions in the active block share local time $u = B(t-1)+b$, which has $\dot{u}^\ell(t) = B$.

Prior Loss The prior loss is unaffected by the block instantiation, and we have

$$\mathcal{L}_{\text{prior}} = \mathrm{KL}\left[q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L}) || p_\theta(\mathbf{z}_1^{1:L})\right] = \sum_{b=1}^{B} \mathrm{KL}\left[q(\mathbf{z}_1^b \mid \mathbf{x}^b) || p_\theta(\mathbf{z}_1^b)\right]. \tag{35}$$

The equality follows from the factorization of both $q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L})$ and $p_\theta(\mathbf{z}_1^{1:L})$ across blocks and the additivity of KL divergence.

Reconstruction Loss For reconstruction, every position in block b completes at global time $(B-b)/B$. At this boundary, the incoming active set is precisely block b, its local time is zero, and its clean context consists of the preceding blocks. Consequently,

$$\mathcal{L}_{\text{recons}}^{\infty,\text{Block}} = \sum_{b=1}^{B} \mathbb{E}_{q(\mathbf{z}^b_{(B-b)/B} | \mathbf{x}^b)} \left[-\log p_\theta\Big(\mathbf{x}^b \mid \mathbf{z}^b_{(B-b)/B}, \mathbf{x}^{<b}\Big)\right]. \tag{36}$$

Diffusion Loss For the diffusion loss, partition the global-time integral into the B active-block intervals and change variables within each interval:

$$u = B(t-1)+b, \qquad du = B\,dt.$$

The factor B from the clock derivative therefore cancels the factor $1/B$ from the integration measure, giving:

$$\mathcal{L}_{\text{diffusion}}^{\infty,\text{Block}} = \sum_{b=1}^{B} \int_0^1 -\frac{1}{2}\mathrm{SNR}'(u) \mathbb{E}_{q(\mathbf{z}^b_{(u+B-b)/B} | \mathbf{x}^b)} \left[\left\| \mathbf{e}^b - \mathbf{e}^b_\theta\left(\mathbf{z}^b_{(u+B-b)/B}, \mathbf{x}^{<b}\right)\right\|_F^2 \right] du, \tag{37}$$

where $\|\cdot\|_F$ represents the Frobenius norm.

Comparing to BD3LM BD3LM (Arriola et al., 2025a) begins from a different perspective, namely that we first factorize the generative model into blocks:

$$p_\theta(\mathbf{x}^{1:L}) = \textstyle\prod_{b=1}^{B} p_\theta\left(\mathbf{x}^b \mid \mathbf{x}^{<b}\right), \tag{38}$$

and then apply diffusion within each block, using the same priors, reverse kernels, and reconstruction decoders. Our Clock Diffusion perspective recovers the BD3LM approach where masked diffusion is replaced with Gaussian corruption.

B.6 Sliding Window Clock Diffusion Language Models

Using the sliding window clocks defined in (11) in Section 3.4, we now derive the corresponding loss terms. Recall that the local clocks are

$$u^\ell_{\text{SW}}(t) := \text{clip}_{[0,1]}\left(\frac{(L+S-1)t + \ell - L}{S}\right), \tag{39}$$

where S is the maximum active-window size. We write $\mathbf{x}^{<\ell} := \mathbf{x}^{1:\ell-1}$ for the clean prefix preceding position ℓ.

Prior Loss Every position still satisfies $u^\ell(1) = 1$. Thus, the forward endpoint distribution and the standard Gaussian prior are identical to those of Block Clock Diffusion, and the prior loss is unchanged:

$$\mathcal{L}_{\text{prior}} = \mathrm{KL}\left[q(\mathbf{z}_1^{1:L} \mid \mathbf{x}^{1:L}) \,\|\, p_\theta(\mathbf{z}_1^{1:L})\right]. \tag{40}$$

Reconstruction Loss Position ℓ completes denoising at global time

$$\tau_\ell = \frac{L-\ell}{L+S-1}. \tag{41}$$

At this boundary, the incoming active set and previously completed positions are

$$\mathcal{A}_{\tau_\ell^+} = \{\ell, \dots, \min(\ell+S-1, L)\}, \qquad \mathcal{C}_{\tau_\ell^+} = \{1, \dots, \ell-1\}. \tag{42}$$

The local times within this incoming window satisfy

$$u^j(\tau_\ell) = \frac{j-\ell}{S}, \qquad j \in \mathcal{A}_{\tau_\ell^+}. \tag{43}$$

Thus, position ℓ has reached local time zero, while subsequent active positions remain at progressively higher noise levels. Substituting these sets into the Clock Diffusion reconstruction loss gives

$$\mathcal{L}_{\text{recons}}^{\infty,\text{SW}} = \sum_{\ell=1}^{L} \mathbb{E}_{\mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell^+}} \sim q(\mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell^+}} | \mathbf{x}^{1:L})} \left[-\log p_\theta \left(\mathbf{x}^\ell \mid \mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell^+}}, \mathbf{x}^{<\ell} \right) \right]. \tag{44}$$

In contrast to Block Clock Diffusion, where reconstruction jointly decodes a block whose positions have all reached local time zero, here, each reconstruction factor decodes one token, conditioned on its clean prefix and an incoming window containing that token and up to $S-1$ partially noisy subsequent positions.

Diffusion Loss Position ℓ is actively denoising over the global-time interval

$$t \in \left(\frac{L-\ell}{L+S-1}, \frac{L+S-\ell}{L+S-1} \right), \tag{45}$$

where its clock derivative is

$$\dot{u}^\ell(t) = \frac{L+S-1}{S}. \tag{46}$$

Substituting this derivative into the general continuous-time diffusion loss gives

$$\mathcal{L}_{\text{diffusion}}^{\infty,\text{SW}} = \frac{L+S-1}{S} \int_0^1 \mathbb{E}_{q(\mathbf{z}_t^{1:L} | \mathbf{x}^{1:L})} \left[\sum_{\ell \in \mathcal{A}_{t^+}} -\frac{1}{2} \text{SNR}'(u^\ell(t)) \left\| \mathbf{e}^\ell - \mathbf{e}_\theta^\ell \left(\mathbf{z}_t^{\mathcal{A}_{t^+}}, \mathbf{x}^{\mathcal{C}_t} \right) \right\|_2^2 \right] dt. \tag{47}$$

The prediction error is evaluated using the sliding window and clean context present along each position's denoising trajectory.

To express this loss in local time, as in the block derivation, we exchange the integral and finite sum and change variables separately for each position:

$$u = \frac{(L+S-1)t + \ell - L}{S}, \qquad dt = \frac{S}{L+S-1}\, du. \tag{48}$$

Each position's active interval maps to $u \in [0,1]$. The clock derivative therefore cancels against the integration measure.

C Additional Training & Evaluation Details for Clock Diffusion Language Models

Throughout this section, we use $\oplus$ to denote concatenation along the sequence axis and $\mathcal{M}_{\text{B}}, \mathcal{M}_{\text{SW}}$ for the block and sliding window training masks of Figure 2, respectively.

Note that the diffusion loss training steps presented in Algorithms 2 and 3 do not depict self-conditioning.

Algorithm 2 Block Clock DLM Training

Require: data $\mathbf{x}^{1:L}$, embedder E, noise schedule γ, forward process q, B blocks of size $S := L/B$, DENOISER with parameters θ, loss $\mathcal{L}$

$t^b \sim \mathcal{U}[0,1], \forall b \in \{1,\dots,B\}$ ▷ one clock per block
$t^\ell \leftarrow t^{b(\ell)}, \forall \ell \in \{1,\dots,L\}$
$\mathbf{e}^{1:L} \leftarrow E[\mathbf{x}^{1:L}]$
$\mathbf{z}_t^{1:L} \sim q(\cdot \mid \mathbf{e}^{1:L}, t^{1:L})$
LOGITS $\leftarrow$ DENOISER$\big(\mathbf{e}^{1:L} \oplus \mathbf{z}_t^{1:L},\ \gamma_0^{1:L} \oplus \gamma_t^{1:L},\ \mathcal{M}_{\text{B}}\big)$
Take a gradient step on $\nabla_{E,\theta}\mathcal{L}(\text{LOGITS}; E, \theta)$

Algorithm 3 Sliding Window Clock DLM Training

Require: data $\mathbf{x}^{1:L}$, embedder E, noise schedule γ, forward process q, window size S, groups $\mathcal{W}_b := \{(b-1)S+1,\dots,bS\}$, DENOISER with parameters θ, loss $\mathcal{L}$

$t^\ell \sim \mathcal{U}[0,1], \forall \ell \in \{1,\dots,L\}$ ▷ one clock per position
$(t^\ell)_{\ell\in\mathcal{W}_b} \leftarrow \text{SORTASCENDING}\big((t^\ell)_{\ell\in\mathcal{W}_b}\big), \forall b$
$\mathbf{e}^{1:L} \leftarrow E[\mathbf{x}^{1:L}]$
$\mathbf{z}_t^{1:L} \sim q(\cdot \mid \mathbf{e}^{1:L}, t^{1:L})$
LOGITS $\leftarrow$ DENOISER$\big(\mathbf{e}^{1:L} \oplus \mathbf{z}_t^{1:L},\ \gamma_0^{1:L} \oplus \gamma_t^{1:L},\ \mathcal{M}_{\text{SW}}\big)$
Take a gradient step on $\nabla_{E,\theta}\mathcal{L}(\text{LOGITS}; E, \theta)$

C.1 Block Training

When using the local clocks from (33) that enable block SAR continuous diffusion, our training algorithm closely resembles that proposed in Arriola et al. (2025a) for block discrete diffusion. For completeness, in Algorithm 2, we present the procedure for computing the diffusion loss. Every position in a block shares a single diffusion time, so we draw B times per sequence, one per block, and broadcast each to the S positions of its block. The mask $\mathcal{M}_{\text{B}}$ is the one depicted in Figure 2: clean-to-clean attention is *block* causal, a noisy query attends to clean keys strictly preceding the start of its own block, and noisy positions attend bidirectionally within their block. Note that for the time conditioning on the clean half of the canvas we use the clean endpoint γ_0.

C.2 Sliding Window Training & Evaluation

In Algorithm 3, we present the method for efficiently computing diffusion loss at each position for Sliding Window Clock Diffusion. As in block training, we divide the noisy latent sequence into disjoint groups of S-width token windows. Each window corresponds to a different awake set. However, the Sliding Window Clock Diffusion training algorithm differs from that of Block Clock Diffusion in two ways: 1) the vectorized training attention mask has a different clean-to-clean attention pattern, and 2) diffusion times need to be unique per position and are not shared within an active window.

Modified Attention Mask Concretely, $\mathcal{M}_{\text{SW}}$ is identical to $\mathcal{M}_{\text{B}}$ except that clean-to-clean attention becomes *token* causal; the noisy-to-clean and noisy-to-noisy quadrants are unchanged, so noisy positions still attend to one another within a width-S window and to clean keys strictly preceding that window.

Non-Uniform Time per Active Window In block training, every position of a block shares one diffusion time, so a single draw per block suffices. In sliding window generation, each position in the active window of S tokens is at a different time in its denoising trajectory. The schedule in (11) would space those times exactly $1/S$ apart, but we deliberately train on *random* spacings instead. We do so to enable Cache Grab samplers (Section 3.6): since these samplers commit a variable number of positions per step, the window advances by a variable stride and adjacent positions can end up separated in time by non-fixed width intervals. We therefore draw L times $t^{1:L} \sim \mathcal{U}[0,1]^L$, one per position, and sort them in ascending order within each of the disjoint length-S groups $\mathcal{W}_b := \{(b-1)S+1,\dots,bS\}$, for $b \in \{1,\dots,L/S\}$. Sorting within groups of S tokens preserves the property that within a sliding window local time increases from left to right.

Proxy Reconstruction Loss for Training The sliding window reconstruction term derived above,

$$\mathcal{L}_{\text{recons}}^{\infty,\text{SW}} = \sum_{\ell=1}^{L} \mathbb{E}_{\mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell}^+} \sim q(\cdot \mid \mathbf{x}^{1:L})} \left[-\log p_\theta\left(\mathbf{x}^\ell \mid \mathbf{z}_{\tau_\ell}^{\mathcal{A}_{\tau_\ell}^+}, \mathbf{x}^{<\ell} \right) \right], \tag{49}$$

asks, for each position ℓ, that the $S-1$ positions after ℓ be held at the *staggered* local times $u^j(\tau_\ell) = (j-\ell)/S$ while ℓ itself sits at local time zero. Each of the L terms in the summation in (49) therefore

imposes a different noise pattern, and evaluating all of them exactly would require L forward passes per sequence.

At training time, we optimize a cheap proxy instead. We place *every* position at the clean endpoint simultaneously, i.e., we set $u^\ell = 0$ for all ℓ, corrupt the whole sequence at $\big(\alpha(0), \sigma(0)\big)$, run a forward pass, plus a bootstrap forward on self-conditioned rows, through the same $2L$ canvas and mask $\mathcal{M}_{\mathrm{SW}}$, and take the token-level cross entropy $-\log p_\theta\big(\mathbf{x}^\ell \mid \mathbf{z}_0^{1:L}, \mathbf{x}^{<\ell}\big)$ at every position. This is the standard clean-endpoint decoder term of a variational diffusion model, and it is evaluated only on the reconstruction subset of the batch selected by (30). The proxy is evaluated under a strictly cleaner context than the one the sampler provides and does not by itself yield a valid bound on $\mathcal{L}_{\mathrm{recons}}^{\infty,\mathrm{SW}}$; we use it purely as a training signal for the clean endpoint.

Reconstruction Loss at Validation *At validation time*, we replace the proxy with an unbiased Monte Carlo estimate of (49). For an offset $\rho \in \{0, \dots, S-1\}$, we partition the sequence into disjoint S-wide groups shifted by ρ and write $j(\ell) := (\ell - \rho) \bmod S$ for the offset of position ℓ within its group. We shift the attention mask with these groups, so each noisy query attends to noisy positions in its own group and clean positions strictly before the group's start. We then corrupt each position at its own local time

$$u^\ell = \frac{j(\ell)}{S}, \tag{50}$$

which reproduces exactly one of the incoming-window patterns from (49): the head of each group (i.e., the position ℓ such that $j(\ell) = 0$) sits at local time zero while subsequent positions in that group sit at progressively higher noise levels. Only the group heads contribute to the cross entropy loss, and their contribution is multiplied by S:

$$\widehat{\mathcal{L}}_{\mathrm{recons}}^{\infty,\mathrm{SW}} := S \sum_{\ell=1}^{L} \mathbf{1}[j(\ell) = 0] \left(-\log p_\theta\big(\mathbf{x}^\ell \mid \mathbf{z}^{1:L}, \mathbf{x}^{<\ell}\big)\right). \tag{51}$$

Note that at sequence boundaries, we retain only in-range positions and include a reconstruction target only if the group head lies within the sequence.

Because ρ is uniform, every position in $\{1, \dots, L\}$ is a group head with probability $1/S$, so multiplying by S exactly offsets the sub-sampling and $\mathbb{E}_\rho\big[\widehat{\mathcal{L}}_{\mathrm{recons}}^{\infty,\mathrm{SW}}\big] = \mathcal{L}_{\mathrm{recons}}^{\infty,\mathrm{SW}}$ for every sequence; averaging over validation batches and devices therefore converges to the exact L-term sum.

Diffusion Loss at Validation The diffusion term requires different noise levels than those used for reconstruction, so we evaluate it separately using a new corrupted input. We keep the same window partition and draw one random number $U \sim \mathcal{U}[0, 1]$ for each window in each sequence. We assign each token the local time

$$u^\ell = \frac{j(\ell) + U}{S}. \tag{52}$$

Within each window, adjacent positions therefore have diffusion times separated by $1/S$. For example, a full window with $S = 4$ has times $U/4, (1+U)/4, (2+U)/4, (3+U)/4$. If the window offset ρ is sampled uniformly, each token is equally likely to occupy any of the S positions within a window. Its local time is consequently uniform on $[0, 1]$, as required for estimating the diffusion integral. We evaluate the diffusion loss at every token, using the weighting from the variational bound.

We use self-conditioning for every validation example: for each loss term, a first model pass produces the full posterior-mean clean embeddings. We feed this prediction into a second pass on the same corrupted input and compute the loss from that second pass. With the single Monte Carlo sample used in our reported evaluations, computing both reconstruction and diffusion therefore requires four model passes per batch.

D Additional Clock Diffusion Sampling Details

Denoising Budgets We briefly comment on what the total number of denoising steps looks like under each of the local clock formulations for block and sliding window clocks.

For block generation, from (33), we have $r = B$, so Algorithm 1 runs for $T = BN$ model calls, with one block awake at any time. Every N steps, an entire block completes in lockstep and is added to the KV cache. So on average we decode S tokens every N steps.

For sliding window generation, from (11), $r = (L + S - 1)/S$, so Algorithm 1 runs for $T = (L+S-1)N/S$ model calls and commits one position every N/S calls, rather than S positions every N as in the block instantiation. Thus, other than at the the end of the sequence, both instantiations commit S tokens every N model calls on average.

Budgets Below the Window Size: $N < S$ The sliding window clocks in (11) offset adjacent positions by $1/S$, while each model call advances every clock by $1/N$. When $N = S$, one position completes per call and Algorithm 1 runs for $T = rN = L + S - 1$ calls. When $N < S$, rN is generally not an integer, so we instead group positions into cohorts of S/N consecutive positions that share one clock. One cohort is admitted per model call and committed together N calls later. The window therefore still holds S positions but advances S/N at a time, and sampling takes $\lceil LN/S \rceil + N - 1$ model calls.

Fused Caching and Denoising As noted in Section 3.2, as written, Algorithm 1 requires an additional model forward to cache decoded tokens. This would be particularly unfavorable for sliding window models that commit a new token to the KV cache on every decoding iteration. We instead *fuse* this caching operation with the subsequent denoising model forward pass by concatenating the newly decoded tokens with the incoming noisy latent active tokens: $\mathbf{x}^{\mathcal{C}_{s|t}} \oplus \mathbf{z}_s^{\mathcal{A}_{s|t}}$. To process this concatenated sequence we then require a custom attention mask for the subsequent denoiser model forward. We depict this mask for both block and sliding window generation in Figure 7.

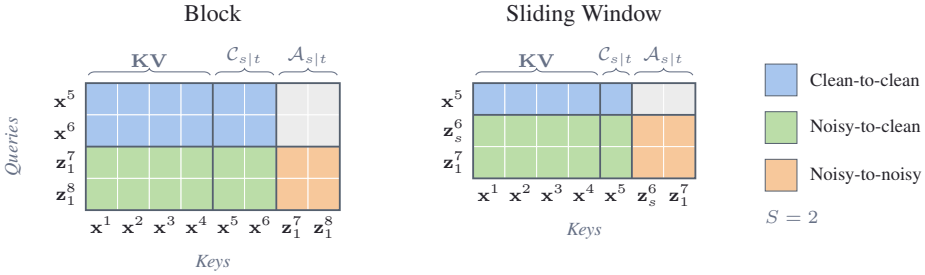


Figure 7: Fused caching and denoising attention masks for (*Left*) block and (*Right*) sliding window generation. Queries are the newly committed tokens $\mathbf{x}^{\mathcal{C}_{s|t}}$ followed by the active noisy tokens $\mathbf{z}_s^{\mathcal{A}_{s|t}}$; keys are the cache $\mathbf{KV}$ followed by the same tokens. Committed tokens never attend to noisy ones, so the keys and values they write to the cache are exactly those a separate caching forward would produce. In this example, we assume the KV cache already has 4 tokens in it. For the block example, we are adding two newly decoded tokens and therefore starting to denoise 2 new samples from the prior on the subsequent denoising pass. For the sliding window example, we are adding one newly decoded token. The window on the subsequent pass consists of the token that was previously in the active window as well as a newly admitted sample from the prior.

E Cache Grab Sampler Algorithms

E.1 Cache Grab-Confidence

Sampling in Window Form Algorithm 1 is written with closed-form clocks: it loops over a fixed grid of $T = rN$ global times and reads the active set off the clocks $u^\ell(t)$. Cache Grab samplers decouple committing from the clocks: a position can be committed before its clock reaches zero, so neither the active sets nor the number of model calls follow the closed-form clocks any more. To detail the Cache Grab sampler algorithms more clearly, we therefore re-write sampling in terms of the state the implementation keeps. Namely, we keep track of an ordered window $\mathcal{A}$ of at most S active positions. For each position $\ell \in \mathcal{A}$, we track its 'age' k^ℓ, which corresponds to the number of denoising steps it has received. We then define the local clock for that position as $u^\ell = 1 - k^\ell/N$. Algorithm 4 presents sampling with this new perspective. Without the Cache Grab samplers and assuming $N = S$, Algorithm 4 is the same as Algorithm 1 for sliding window ClockDLMs.

Algorithm 4 ClockDLM Sampling in Window Form (Sliding Window, $N = S$)

Require: prompt, length L, window size S, per-position budget $N = S$, prior q_{prior}, DENOISER, SAMPLER, commit rule COMMIT

$\mathbf{KV} \leftarrow \text{CACHE}(\text{prompt}, \emptyset)$
$\mathcal{A} \leftarrow \emptyset$ ▷ $\mathcal{A}$: tracks active positions, left to right
$f \leftarrow 0$ ▷ tracks number of model calls
$p \leftarrow 1$ ▷ next position to admit
while $\mathcal{A} \neq \emptyset$ or $p \leq L$ **do**
  while $|\mathcal{A}| < \min(f + 1, S)$ and $p \leq L$ **do** ▷ refill; capacity grows by one per step until it reaches S
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{p\}$
    $\mathbf{z}^p \sim q_{\text{prior}}$
    $k^p \leftarrow 0$
    $p \leftarrow p + 1$
  end while
  $\text{LOGITS}^{\mathcal{A}} \leftarrow \text{DENOISER}(\mathbf{z}^{\mathcal{A}}, u^{\mathcal{A}}, \mathbf{KV})$ ▷ $u^\ell = 1 - k^\ell/N$
  $\mathbf{z}^{\mathcal{A}} \leftarrow \text{SAMPLER}(\text{LOGITS}^{\mathcal{A}})$
  $k^\ell \leftarrow k^\ell + 1$ for all $\ell \in \mathcal{A}$
  $\mathcal{C} \leftarrow \text{COMMIT}(\mathcal{A}, \text{LOGITS}^{\mathcal{A}})$ ▷ under standard decoding, this amounts to: $\{\ell \in \mathcal{A} \mid k^\ell = N\}$
  $\mathbf{x}^{\mathcal{C}} \leftarrow \arg\max(\text{LOGITS}^{\mathcal{C}})$
  $\mathbf{KV} \leftarrow \text{CACHE}(\mathbf{x}^{\mathcal{C}}, \mathbf{KV})$
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{C}$
  $f \leftarrow f + 1$
end while
return $\mathbf{x}^{1:L}$

Cache Grab-Confidence replaces the commit rule of Algorithm 4 with Algorithm 5. A position is eligible to be committed if its clock has run out, as in standard decoding, or if the probability the model assigns to its $\arg\max$ token is at least a threshold η. The committed set is the longest run of eligible positions starting from the left end of the window, so even if some position ℓ clears the η threshold, it may not be committed if a different position $\ell' < \ell$ does not meet the confidence threshold and its local clock has not run out (see, for example, Figure 3, *Left*). After we commit the leftmost tokens, the remaining positions keep their latents and ages and shift left, and the next iteration refills the window with samples from the prior. With $\eta = 1$ the confidence test is disabled and the standard sampler is recovered.

Algorithm 5 Cache Grab-Confidence Commit Rule

function $\text{COMMIT}(\mathcal{A} = (\ell_1, \ldots, \ell_m), \text{LOGITS}^{\mathcal{A}}; \text{threshold } \eta)$
  for $i = 1, \ldots, m$ **do**
    $e_i \leftarrow \mathbf{1}\left[k^{\ell_i} = N\right] \vee \mathbf{1}\left[\eta < 1 \wedge \max_v \text{softmax}(\text{LOGITS}^{\ell_i})_v \geq \eta\right]$ ▷ eligible
  end for
  $j \leftarrow \max\{\, i \in \{0, \ldots, m\} \mid e_1 = \cdots = e_i = 1 \,\}$ ▷ longest eligible prefix
  return $\{\ell_1, \ldots, \ell_j\}$
end function

Adapting Cache Grab-Confidence for Block Decoding B-ClockDLMs denoise a whole block in lockstep and commit it at once, so the rule is applied to the block as a unit. After each model call, if every position being generated in the current block reaches the threshold η, the block is committed; otherwise the block takes its next denoising step, and after its N-th call it is committed. The committed block is written to the cache in the next block's first model call, with full attention within the block (Figure 7, *Left*), and the next block starts with samples from the prior.

E.2 CACHE GRAB-SSD

In Cache Grab-SSD, each iteration makes two model calls. The first is the usual denoising call over the window, from which every active position receives a draft token. The second call runs the model in AR mode over the drafts, attending causally among the draft tokens. At every position in the draft window, the AR head provides its prediction given the cache and draft tokens to its left. The sampler commits the longest prefix of the window on which the draft and the AR prediction

agree, together with the AR prediction at the first position where they disagree. Thus every iteration commits between 1 and $|\mathcal{A}|$ tokens. A position's clock running out never commits it: a position that reaches $u^\ell = 0$ stays there and is drafted again until it is committed, as an accepted draft or as the AR correction. This maintains the invariant that the produced sequence matches the one we would see under AR decoding alone.

Keeping Rejected Positions Self-speculative decoding in discrete diffusion discards the positions it rejects. Cache Grab-SSD instead keeps every position it does not commit, together with its denoising progress.

Staggering Noise in the New Active Window At the start of each iteration, the window is refilled to S positions. Newly admitted positions start from the prior, at $u^\ell = 1$. Positions kept from earlier iterations advance by one step per iteration, $u^\ell \leftarrow \max(u^\ell - 1/N, 0)$. To maintain staggered noise in the active window, if the newly admitted positions are not committed via SSD, their local clocks are spread evenly over the gap between the newest position kept from earlier and $1 - 1/N$. This ensures that the window stays ordered from clean on the left to noisy on the right. Concretely, if u_b is the new time of the newest position kept from earlier ($u_b = 0$, if no position from earlier remains in the window), the i-th of n surviving new positions, counted from the left, is placed at $u^\ell = u_b + \frac{i}{n}(1 - 1/N - u_b)$; the rightmost therefore lands at $1 - 1/N$.

F Additional Experimental Setup Details

Optimization All models trained in our experiments use AdamW (Kingma et al., 2014; Loshchilov et al., 2017). We clip gradients to a maximum norm of 1.0 and use a constant learning rate after linear warm-up. For RePlaid and ClockDLMs (as well as B-ClockDLMs), the learnable embedding matrix has a learning rate 10^{-2}, while the latent, self-conditioning and time-conditioning input projections have a learning rate 3×10^{-4}; these parameter groups have zero weight decay. We maintain an exponential moving average (EMA) of the weights with decay 0.9999 and use these EMA weights for evaluation. The remaining optimizer settings, batch sizes and training budgets are listed in the task-specific tables.

Numerical Precision All models trained in our experiments, including MDLM, BD3LM and SetDLM, use `bfloat16` autocast for the Transformer blocks and output head, including the vocabulary projection. Layer normalization and RMS normalization compute their statistics in `float32`. Validation loss sums and token counts are accumulated and reduced across devices in `float64` before computing perplexity. TF32 is enabled for eligible `float32` matrix multiplications.

For ClockDLMs and our RePlaid baselines, embedding normalization, latent corruption, the latent and self-conditioning input projections, time embeddings, posterior-mean embedding computations and the analytic output logit prior use `float32`. Their learned noise schedules are evaluated in `float64`. For their output logit prior in (31), we clip $(\sigma_t^\ell)^2$ to a minimum of 10^{-8}.

Evaluation Protocol Our evaluations run on a single node with eight `H100-80GB` GPUs, using seed $= 1$. Generation uses batch size one per GPU; likelihood evaluation uses the batch size specified below. Our NFE metrics account for all model forwards, including prompt ingestion, cache updates, final readout calls, and AR verification calls for Cache Grab-SSD. Sampling algorithms, denoising budgets and stopping rules are specified below for each task.

Distinguishing Cache Grab-SSD All of the results for ClockDLM represent standard diffusion. Any results reported for Cache Grab-Confidence use these models as well. To enable Cache Grab-SSD, we **retrain** separate models with the AR loss enabled. Only results that are explicitly marked as using Cache Grab-SSD, as well as those in Table 8, use models trained with joint AR and diffusion loss.

F.1 OpenWebText

Data We train on OpenWebText (Gokaslan et al., 2019).[1] Following Sahoo et al. (2024a); Arriola et al. (2025a), the last 100,000 documents of the `train` split are held out for validation and the remainder is used for training. Documents are tokenized with the GPT-2 tokenizer (Radford et al.,

[1] https://huggingface.co/datasets/Skylion007/openwebtext

2019). We append [EOS] to each document and concatenate the tokenized documents within each pre-processing batch. The stream is split into chunks of $L-2$ tokens, each wrapped with [BOS] and [EOS] to form a sequence of length $L = 1024$. Incomplete chunks at the end of each pre-processing batch are discarded; the resulting sequences contain no padding.

Training For our retrained RePlaid model and ClockDLMs we learn both the noise schedule γ_t and the endpoints γ_0 and γ_1, following the setup in Yang et al. (2026). We initialize the endpoints to $\gamma_0 = -5.5$ and $\gamma_1 = 3$. We use the same monotone MLP as in Kingma et al. (2021); Yang et al. (2026) for the learned schedule. Full training details are provided in Table 4.

Likelihood Evaluation We report variational perplexity upper bounds on the full held-out validation split, with validation batch size 32 per GPU. RePlaid and Block ClockDLMs use the ordinary validation negative log-likelihood bound, while Sliding Window ClockDLMs use the rolling estimator described in Appendix C.2. We evaluate reconstruction and diffusion separately, using self-conditioning for every validation example. All evaluations use the 1,000,000-step checkpoint and EMA weights.

Sampling Evaluation To produce the generative quality results in Appendix G.1, we use a DDPM sampler. We also explore nucleus (top-p) filtering: at every denoiser call, we keep the smallest set of most probable tokens whose cumulative probability is at least p, re-normalize over them, and use the expected embedding under this truncated distribution as the clean-embedding estimate in the DDPM update.

Table 4: Experimental configuration for OpenWebText.

Data	
Tokenizer / vocabulary	`gpt2` / 50,258
Context length L	1024
Packing	[EOS] per document; [BOS]/[EOS] per packed block
Validation split	Last 100,000 documents
Architecture	
Backbone	DiT (Peebles et al., 2023) with adaptive layer norm
Layers / hidden / heads	12 / 768 / 12 (head dim. 64)
Conditioning dim. / dropout / MLP ratio	128 / 0.1 / 4
Embedding bottleneck d	16 (L_2-normalized rows, untied output head)
Optimization	
Gradient updates	1,000,000
Global / per-device batch size	512 / 8 (no gradient accumulation)
GPUs	64 (8 nodes $\times$ 8 `H100-80GB`)
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, wd 0
Learning rate / warm-up	3×10^{-4}, constant after 2,500 linear-warmup steps
Embedding / projection LR	10^{-2} / 3×10^{-4} (wd 0)
Gradient clipping / precision / EMA	1.0 / `bf16` / 0.9999
Time sampling	Antithetic, stratified over the global batch
Clock Diffusion	
Window sizes S	$\{4, 8, 16, 32, 64\}$
γ_t parameterization	Monotone MLP (Kingma et al., 2021), hidden width 1024
$[\gamma_{\min}, \gamma_{\max}]$ (initialization)	$[-5.5, 3.0]$, both endpoints learned
Schedule LRs (shape / endpoints)	10^{-2} (wd 0) / 10^{-2} (wd 0.1)

F.2 Math Reasoning Benchmark

Data We train on TinyGSM (Liu et al., 2023),[2] a synthetic corpus of grade-school math questions paired with Python programs whose execution produces the answer. Each example is tokenized as [BOS] $\oplus$ question $\oplus$ \n $\oplus$ code $\oplus$ [EOS]; the prompt ([BOS], question and \n) is provided as clean conditioning context and diffusion is applied to the code continuation. Examples exceeding the $L = 512$ canvas are discarded and shorter examples are right-padded. For the diffusion models,

[2] https://huggingface.co/datasets/TinyGSM/TinyGSM

we include padded positions in the loss, as we found that this led to improved performance. We evaluate zero-shot on the full GSM8K (Cobbe et al., 2021) test set (1,319 examples).[3]

Training For our retrained RePlaid model and ClockDLMs we learn the noise schedule γ_t and keep endpoints fixed at $\gamma_0 = -1$ and $\gamma_1 = 6$. We use the monotone log-slope schedule model described in Appendix B.4. Full training details are provided in Table 5.

Evaluation Each configuration decodes all 1,319 test problems once with seed 1 and batch size one per device. When applicable, we apply early stopping on `[EOS]`. For example, full sequence MDLM and RePlaid cannot benefit from this early stopping. We follow the evaluation protocol from Deschenaux et al. (2026). A generated output is scored by extracting the emitted program and executing it. The returned value is compared against the reference answer with a tolerance of 10^{-3}. We report pass@1 accuracy. All evaluations use the 250,000-step checkpoint with EMA weights.

Sampling RePlaid, Block ClockDLMs and Sliding Window ClockDLMs use DDPM ancestral sampling with vocabulary top-1 at every denoising update, meaning that we sample $\mathbf{z}_s \sim q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \arg\max(\mathbf{x}_\theta))$, i.e., each DDPM update therefore uses the embedding of the highest-probability token as its clean-embedding estimate. The results in Table 2 reflect $N = S$ for our models. For Block and Sliding Window ClockDLMs we sweep the per-position budget $N \in \{S, S/2, \ldots, 2\}$ to produce the accuracy–efficiency trade-offs in Figure 4. RePlaid is decoded full-sequence with $T \in \{2, 4, \ldots, 1024\}$. The SW-SetDLM curves in Figure 4 instead sweep the confidence threshold. SAR MDLM decoding refers to the sampling employed by models such as LLaDA (Nie et al., 2025), where at each denoising iteration, only positions from a leftmost block of size S are eligible for decoding. For BD3LM and MDLM, we use confidence-based decoding, where the top-k most confident masked tokens are generated on every iteration. We control the denoising budget by varying k. Cache Grab-Confidence evaluations use $N = S$, with the early-commit threshold swept over $\{0.99, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7\}$. Cache Grab-SSD runs at $N = S$. For self-conditioning, we use the top-1 embedding from the previous denoising iteration.

F.3 Summarization

Data We use CNN/DailyMail v3.0.0 (Hermann et al., 2015; See et al., 2017).[4] Following Arriola et al. (2026), the source region is `[BOS]` $\oplus$ article $\oplus$ `[EOS]`, independently truncated to 512 tokens, and the target region is `Summary:` $\oplus$ highlights $\oplus$ `[EOS]`, independently truncated to 180 tokens; both regions reserve their final position for `[EOS]` and are concatenated and right-padded to the $L = 768$ max sequence length. The article region is provided as clean conditioning context. We only noise and compute the diffusion loss and the reconstruction loss on the target region, and padded positions are excluded from both the loss and the attention mask. We select the checkpoint with the lowest validation loss for evaluation.

Training For our retrained RePlaid model and ClockDLMs we learn the noise schedule γ_t and keep endpoints fixed at $\gamma_0 = -5$ and $\gamma_1 = 3$. We use the monotone log-slope schedule model described in Appendix B.4. Full training details are provided in Table 6.

Evaluation Each configuration generates summaries for all 11,490 test articles once with seed 1 and batch size one per device. We adopt the decoding recipe of Arriola et al. (2026): at most 180 new tokens are generated, with a minimum of 50 non-`[EOS]` tokens generated. We apply an exponential-decay length penalty with a factor 1.1 that starts after 80 tokens, and a repetition penalty swept over $\{1.0, 1.2\}$. Logits processors are applied in a fixed order: log-softmax, length penalty, repetition penalty, renormalization. Where applicable, we use early stopping on the `[EOS]` token. We report ROUGE-1/2/L (Lin, 2004) computed with the TorchMetrics implementation without stemming and averaged over examples.

Sampling We use the same sampling setup as in the GSM8K experiment. RePlaid, Block ClockDLMs and Sliding Window ClockDLMs use DDPM ancestral sampling with vocabulary top-1 at every denoising update (as in the GSM8K setup). Fixed conditioning tokens use their known clean embeddings wherever self-conditioning is supplied. The per-position budget is swept over $N \in \{S, S/2, \ldots, 2\}$ and the Cache Grab-Confidence threshold sweep is run at $N = S$; RePlaid is decoded full-sequence.

[3] https://github.com/openai/grade-school-math

[4] https://huggingface.co/datasets/abisee/cnn_dailymail

Table 5: Experimental configuration for TinyGSM training and GSM8K evaluation.

Data	
Tokenizer / vocabulary	`HuggingFaceTB/SmolLM-135M` / 49,153
Context length L	512 (right-padded, overlong examples filtered)
Packing	None; one example per sequence
Prompt / target	`[BOS]`$\oplus$question$\oplus$`\n` / code$\oplus$`[EOS]`
Validation split	1% deterministic holdout of TinyGSM train
Architecture	
Backbone	DiT, identical to Table 4
Layers / hidden / heads	12 / 768 / 12
Embedding bottleneck d	16
Optimization	
Gradient updates	250,000 (all reported checkpoints)
Global / per-device batch size	512 / 16 (no gradient accumulation)
GPUs	32 (4 nodes $\times$ 8 `H100-80GB`)
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, wd 0
Learning rate / warmup	3×10^{-4}, constant after 2,500 linear-warmup steps
Gradient clipping / precision / EMA	1.0 / `bf16` / 0.9999
Time sampling	Antithetic, stratified over the global batch (except SetDLM)
Validation / checkpoint interval	every 10,000 steps (500 validation batches)
Clock Diffusion	
Window sizes S	$\{4, 8, 16, 32, 64\}$
γ_t parameterization	Monotone log-slope, 32 knots, shape only
$[\gamma_{\min}, \gamma_{\max}]$	$[-1, 6]$, endpoints frozen
Schedule shape LR	10^{-4} (wd 0)
Cache Grab-SSD models	AR head tied to the diffusion output layer, loss weight 1.0
Evaluation	
Test set	GSM8K test, all 1,319 examples, seed 1
Sampler	DDPM, top-1 at each update, EOS stopping
Denoising budget	$N \in \{S, S/2, \ldots, 2\}$
Confidence thresholds	$\{0.99, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7\}$ at $N = S$

G Additional Experimental Results

G.1 Open Ended Generation

Using our ClockDLM ($S = 64$) model and the retrained RePlaid from the OpenWebText experiment, we evaluate the quality of generated samples. Specifically, we follow the setup from Yang et al. (2026), where we generate 5,120 sequences and use ModernBERT-Large (Warner et al., 2025) to compute MAUVE scores (Pillutla et al., 2021; Pillutla, 2021). For both models we sweep over using a repetition penalty with factor 1.2 and applying top-p nucleus sampling to the softmax output of the denoising network with $p \in \{0.9, 0.95\}$. For RePlaid, we consistently find best results using no nucleus sampling and no repetition penalty. We found ClockDLM to work consistently better when using the repetition penalty. However, our model was more sensitive to the top-p parameter with improved results for $p = 0.95$ when using NFEs $> L$ and improved results for $p = 0.9$ when using an NFE budget $< L$.

In Figure 8, we display the best results for each model. Notably, in higher NFE regimes $\sim 512+$, ClockDLM outperforms RePlaid, whose performance plateaus and even dips after a certain budget. However, for this task of open-ended generation in the low-NFE regime, our model struggles. This can be attributed to the fact that in this regime we apply a comparatively much smaller denoising budget per token. For example, with a total denoising budget of ~ 256, ClockDLM will apply ~ 16 denoising steps per token, whereas RePlaid, because it is a full sequence model, uses this same total decoding budget for each token, applying 256 denoising steps to each token.

Table 6: Experimental configuration for CNN/DailyMail.

Data	
Tokenizer / vocabulary	`Qwen/Qwen3-0.6B-Base` / 151,669
Context length L	768 (512 source + 180 target, right-padded)
Target prefix	`"Summary: "`
Supervision	Target region only; padding neither noised nor scored
Architecture	
Backbone	Qwen3-style decoder (architecture only; trained from scratch)
Layers / hidden / SwiGLU intermediate	28 / 256 / 768
Query / key-value heads, head dim.	16 / 8, 128
Normalization / positions	RMSNorm ($\epsilon = 10^{-6}$, per-head Q/K norm) / RoPE $\theta = 10^6$
Conditioning dim. / biases / tying	128 / none / untied
Embedding bottleneck d	16
Optimization	
Gradient updates	$\leq$ 500,000; best-validation checkpoint selected
Global / per-device batch size	128 / 4
GPUs	32 `H100-80GB`
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.98)$, $\epsilon = 10^{-8}$, wd 10^{-5}
Learning rate / warmup	3×10^{-4}, constant after 1,000 linear-warmup steps
Gradient clipping / precision / EMA	1.0 / `bf16` / 0.9999
Time sampling	Uniform (antithetic sampling disabled)
Validation / checkpoint interval	every 5,000 steps
Clock Diffusion	
Window sizes S	$\{4, 8, 16, 32, 64\}$
Endpoint terms	$\mathcal{L}_{\text{recons}}$ optimized (weight 1.0); $\mathcal{L}_{\text{prior}}$ evaluated only
γ_t parameterization	Monotone log-slope, 32 knots, shape only, frozen endpoints
$[\gamma_{\min}, \gamma_{\max}]$	$[-5, 3]$, endpoints frozen
Evaluation	
Test set	CNN/DailyMail test, all 11,490 examples, seed 1
Max. / min. generated tokens	180 / 50 non-`[EOS]`
Length penalty (start, factor)	(80, 1.1)
Repetition penalty	$\{1.0, 1.2\}$
Metric	ROUGE-1/2/L, TorchMetrics, no stemming, example mean

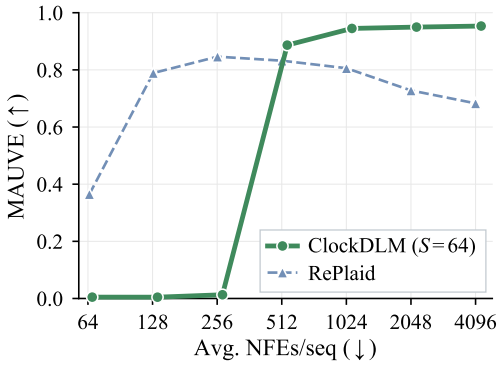


Figure 8: MAUVE scores vs. NFEs/seq for sequences generated from models trained on OWT.

G.2 Cache Grab-Confidence for ClockDLMs & B-ClockDLMs on GSM8K

Table 7 reports the full Cache Grab-Confidence threshold sweep on GSM8K for ClockDLMs and B-ClockDLMs at $S \in \{32, 64\}$. For ClockDLMs, lowering the threshold to $\eta = 0.7$ reduces NFEs/seq by 3.2× at $S = 32$ with no loss in accuracy, and by 2.8× at $S = 64$ at a cost of 2.0 points. B-ClockDLMs, which commit a block only once every position in it clears the threshold, see smaller reductions of 2.7× at $S = 32$ and 2.1× at $S = 64$, with accuracy within one point of

standard decoding in both cases. Lowering the threshold produces the expected efficiency gains of lowering NFEs / sequence. However, interestingly, we do not see a clear monotonic trend in accuracy reduction as the threshold is reduced. This is likely due to the nature of this task's evaluation metric: correctness of the executed program that is generated can be significantly altered with even small decoding deviations.

Table 7: Cache Grab-Confidence on GSM8K. Zero-shot pass@1 accuracy (Acc %; ↑) on the GSM8K test set and NFEs/seq for ClockDLMs and B-ClockDLMs trained on TinyGSM, sweeping the early-commit threshold η at $N = S$. NFE values are means ± sample standard deviations. *Standard* rows decode without early commits.

		$S = 32$		$S = 64$	
		NFEs/seq. (↓) Mean$_{\pm SD}$	Acc. % (↑) 0-shot, pass@1	NFEs/seq. (↓) Mean$_{\pm SD}$	Acc. % (↑) 0-shot, pass@1
B-ClockDLM					
Standard		$190.1_{\pm 47.3}$	54.3	$220.1_{\pm 44.3}$	49.1
+ Cache Grab-Conf.	($\eta = 0.99$)	$134.8_{\pm 37.6}$	51.6	$182.3_{\pm 45.1}$	48.5
	($\eta = 0.95$)	$117.5_{\pm 39.8}$	53.8	$161.8_{\pm 47.0}$	47.2
	($\eta = 0.9$)	$105.1_{\pm 38.3}$	53.0	$146.5_{\pm 49.1}$	48.0
	($\eta = 0.85$)	$95.6_{\pm 37.4}$	**54.7**	$135.8_{\pm 49.6}$	49.1
	($\eta = 0.8$)	$86.8_{\pm 36.6}$	53.6	$125.8_{\pm 50.2}$	**49.4**
	($\eta = 0.75$)	$79.0_{\pm 35.7}$	51.6	$113.1_{\pm 48.1}$	48.6
	($\eta = 0.7$)	$71.1_{\pm 33.7}$	53.4	$103.6_{\pm 48.3}$	48.4
ClockDLM					
Standard		$188.1_{\pm 44.8}$	59.6	$219.1_{\pm 43.4}$	**55.6**
+ Cache Grab-Conf.	($\eta = 0.99$)	$123.0_{\pm 35.1}$	57.7	$161.5_{\pm 41.5}$	51.6
	($\eta = 0.95$)	$105.6_{\pm 36.0}$	57.6	$139.3_{\pm 41.0}$	52.7
	($\eta = 0.9$)	$92.8_{\pm 34.5}$	58.3	$123.3_{\pm 40.1}$	54.9
	($\eta = 0.85$)	$82.3_{\pm 32.0}$	58.8	$112.2_{\pm 40.4}$	55.0
	($\eta = 0.8$)	$74.6_{\pm 31.0}$	**60.0**	$99.9_{\pm 40.0}$	53.0
	($\eta = 0.75$)	$67.1_{\pm 29.8}$	59.3	$88.7_{\pm 38.1}$	55.0
	($\eta = 0.7$)	$59.5_{\pm 27.6}$	**60.0**	$78.4_{\pm 35.6}$	53.6

G.3 Effect of SSD Training

Table 8 isolates the effect of the joint AR and diffusion training objective used to train models that enable Cache Grab-SSD. All models are decoded with the standard diffusion sampler, i.e., without SSD. Joint training improves accuracy at every window size, by 0.9 to 5.7 points, while leaving NFEs/seq essentially unchanged. At $S \in \{4, 8\}$, the jointly trained models (66.6% and 67.2%) even exceed the AR baseline in Table 2 (64.3%).

Table 8: Effect of joint AR and diffusion loss training. Zero-shot pass@1 on all 1,319 GSM8K test examples for Sliding Window ClockDLMs trained on TinyGSM, decoded with the standard diffusion sampler in both cases. Adding the AR loss leaves the denoising budget unchanged but improves accuracy at every active window size S. NFEs are means with ± sample standard deviations in subscripts.

	Diffusion loss only		Joint AR + diffusion loss		
S	NFEs/seq. (↓)	Acc. % (↑)	NFEs/seq. (↓)	Acc. % (↑)	Δ
4	$161.5_{\pm 46.5}$	65.7	$160.4_{\pm 45.6}$	**66.6**	+0.9
8	$165.0_{\pm 45.8}$	63.8	$164.5_{\pm 45.6}$	**67.2**	+3.4
16	$172.6_{\pm 44.9}$	60.0	$173.7_{\pm 47.0}$	**63.8**	+3.8
32	$188.1_{\pm 44.8}$	59.6	$187.4_{\pm 44.4}$	**63.5**	+3.9
64	$219.1_{\pm 43.4}$	55.6	$219.5_{\pm 44.3}$	**61.3**	+5.7

G.4 Generated Sample Examples

We present selected generations from the saved evaluation runs, using the model checkpoints and sampling procedures described above. Examples are screened for non-explicit content; the GSM8K examples are additionally checked for correct program logic and final answers. Generated wording is preserved, and the CNN/DailyMail comparisons use the same articles for both models. All runs use EMA weights and base seed 1.

OpenWebText We show complete $1{,}024$-token generations from two configurations in Figure 8: RePlaid with $T = 256$, no nucleus truncation and no repetition penalty, and ClockDLM with $S = N = 64$, top-$p = 0.95$ and repetition penalty 1.2. Both use DDPM at temperature 1 and reuse the preceding clean-embedding estimate for self-conditioning: the full posterior mean for RePlaid and the nucleus-truncated posterior mean for ClockDLM. The respective numbers of executed model forwards are 257 and $1{,}087$ per sequence.

RePlaid — OpenWebText

<|endoftext|> few days. Then seabirds who had died on the surface threw off the rafts and carcasses. Seabirds started diving for food But the rushing cold water worsened the situation and released the polar chicks, as they apparently did sometimes. The fact that it was possible in such cold waters for plankton chicks to be preserved, says Sabulis, shows that Antarctic animals were not getting along well. In fact, the female plankton, Cepolina, itself died just after its arrival, because of freezing during the warm winter. Later on, heavy meltwaters flooded the Columbia River on the northern coast of Antarctica, and began to wash away the chicks. Along with them were the bodies of two clams, Dionosaurus and Vesiformes, which had hatched. "Scientists did not think," says Sabulis, that Cepolina was made of this different mass and could therefore be used for cod. However, with some clams that remained alive, the warmer water suggests that their deaths were caused by traffic in the river. Other clams remain in the lake, along with many juveniles, and Sabulis says they suggest that the body from Cepolina was probably used for milking up seabirds (or river pancreas). "Milking on such a large volume of land — consisting largely of fish," says Sabulis — "makes it even harder for salmon and rivermen to maintain a better economic standard of living than minks, humpback turtles and fish." [3]
See also [edit]<|endoftext|>Donald Trump — an unconventional presidential candidate — rallied a group of business people in the state's capital last week. Though mostly miners and traditional coal industry workers, dozens of Trump supporters were ranging from mine owners to port builders and employee government workers. In recent weeks, Trump has shown the heavy weight of business and commerce, spotlighting government regulations and thrusting job creation into a top national priority.
On the grounds of Idaho Lutheran Brothers High School Chapel in Boise, Republican candidate Donald Trump announces his intention to win the Oregon Republican nomination for president onApril 16, 2016. (Photo: The Oregonian)
"A lot of the miners he's spoken against are men," said Greg Summer, a Trump supporter who attended a meeting here. "They don't get any healthcare. They get no family."
Trump's 'teaching message' puts Westminster in jeopardy
County public sector workers said they value their candidate's authenticity.
"We think he is a great voice for people, both for good working conditions and small business and applaud his insistence on giving them time off work," Rob Solomon, national director of Local 1148, a public sector union in Gadsden, Ore., said in an email.
But others on the business side of the stump expressed a greater acceptance of Trump's message. Both decent working conditions and small businesses are important, said Jim Frankel, the owner of a brick-and-mortar outdoor store in Lincoln Village. He also serves as a vice chairman of the union-backed group Coalition for American Jobs and acknowledges that those issues "belong to some of the small business owners" who support Trump. But good working conditions remain, he said, "and certainly we accept his message."
Critics see huge risks in a Trump battleground economy, especially compared with the businessman outlining explicitly his plan to expand job creation. Frank Tucker, an adviser and consultant at BRCA Consulting LLC, a coal-industry watchdog, has criticized that approach as generally unswerving. Trump's first jobs plan shows a thrashy approach and could be a significant slog, he said.
"The coal recovery is arguably more important not only because of his small business investments, but in part because of his repudiation of core conservative policies," Tucker said. "Trump is arguably the most popular union and small business candidate in the race and has consistently proposed developing infrastructure projects that would lock in any permanent coal jobs."
Other supporters argue that politics is about big business. Jeff Martinez, a Trump supporter in Glenport, Wash., said he has supported coal policies championed by Hillary Clinton and wants to see her "doing this our own way." When he asked if Democratic nominee Clinton supported coal workers in communities such as Rampunda Range, Ramos said, "Hillary is very big on personal money."
County business owners say Trump is the only candidate for president emphasizing conservative government rules that can actually spur job creation.
Why Americans most needed backing
At the Trump rally, miners from King counties, Spokane, Wash., Vernon and others in Oregon were waving small signs and posters related to their work. Some were unions, wearing a tinted coat and a painted hat on a baseball bat. Some had been inspired by "Manchita Run," the popular calendar from 1900s in Oregon.
Last year, when labor activists in Boise urged the Idaho state legislature to strike down a law on miners' rights, a Walmart store in Mont<|endoftext|>

Figure 9: Full OpenWebText generation from RePlaid, saved sequence index 0, without nucleus truncation or a repetition penalty.

ClockDLM — OpenWebText

<|endoftext|> round up employees who worked abroad to write the rules.
The rules he quietly signed were for these in 2010, under the tweaking of the Competition Act of 2005, a law for companies that deal with international trade that went into one constitutional term in 2007 and was repealed in 2014.
The enthusiasm for limits on the vulnerable often reminds me of the time you look for a train from the opposite end, leaving the station in time to witness the aspirations of your mates, friend your parents, examine the new gear, focus your eyes on the approaching fireworks. When someone wants to embrace the challenge of the job, nothing about these rules will do.<|endoftext|>This was a dream of mine, Erica's queue broadcaster that can broadcast messages to a channel or a console, and can push notifications whenever necessary. Sorry if you missed this and see below:
So, let's continue down this road!
The Objective
I created this "Condition Scaling Script". All it has to do is tell you you may have the value processing defined, and you can type in the message or scroll it with your finger.
For example, you might type this:
yourMessage.hasValue(0, 0);
Which is not only to your message, but your Messenger…
What's Next?
In this post, I'll be explaining the concepts of interface and access in a nutshell by telling you what I managed. If your knowledge becomes too limited, it might be time to dive deeper and explore some tips that to make the path easier.
To Get You Started With It
You need to set document.keyText.hasValue :
class Window { string idText; string prevText; window.setIgnoringText(idText); }
When your Window gets too big, like this one, prevText will take this string, and then try to copy the contents you read onto the current session.
And you should also set an prevText value of zero, so the clipboard can only hold the text it currently has.
Knowing this allows you to use it globally. Just like when you click the Bookmarks button everywhere you go, you can easily get the list of all messages. That's similar to the waterfall on the splash screen, and you can easily create a variety of different kinds of events to override displaying the content if needed.
The Internet Explorer is burdened with a screen view option. This allows you to fearlessly know where your messages come from in more than one screen view, and then not geolocate your friends. Great. Right now, the program can replay itself 2 million words at a time, and to date it has been broadcasting messages in multiple screen views, as one action and for multiple users, at the same time window it is reading.
When your Window is too big, it will copy all messages in the Internet Explorer Frame, which causes it globally.
How to Use It
Since the program returns the list of messages consumed by the user, it can change the screen top display width to its own depurion. And you can use a method like store() , which copies the messages emitted without being viewed.
More beautiful crummy interface.
Conclusion
I got beautiful results sticking to the "Condition Scaling" script alone, all the way until I finally executed it, a bunch of data from many people come and went, and hours spent on calls, processing messages across many different mouse sessions. And finally, once it's finished, everyone has had a blast using it.
This is definitely a milestone for lots of other designers, who need to put together a useful program, and have feature-rich interactions that they can't seem to afford, let alone writing a beautiful piece of code. And I hope that many of you will love that.<|endoftext|>DemTV Live 05/05/2017 (Update 8)
Live Online Android streaming service has a third major update that has been recently pushed to DemTV Live (Update 4). This update covers several new features. The main part of the update is the ability to stream streams of any seasons via the official Netflix or Google Play TV sites. The purpose of the streaming builds upon the current DemTV Live website, is to allow viewers to watch their most-anticipated seasons in 4K HD. Here is from the description of the update:
"High Quality™ Streaming (Premium Premium) is designed to deliver gorgeous photos and an affordable way to experience almost anything. The new High Frame Streaming (HDDR™) interface can accommodate up to 1 gigabyte of raw or native video. Additional frame sizes provide extra Turbo HD Memory capabilities sufficient to handle up to 6 gigabytes in native video to deliver 100% high quality and immersive video. This change builds to DemTV Live's future strategy of high image quality and epic video delivery<|endoftext|>

Figure 10: Full OpenWebText generation from ClockDLM, saved sequence index 1, with top-$p = 0.95$ and repetition penalty 1.2.

GSM8K These ClockDLM examples use $S = N = 32$, DDPM with top-1 clean-token predictions and token-embedding self-conditioning, and the $250{,}000$-step EMA checkpoint. The displayed results are those recorded by the evaluation program.

ClockDLM — GSM8K ($S = N = 32$, 198 NFEs)

Question
Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

Generated program

```
def simple_math_problem() -> int:
    """
    Janet's ducks lay 16 eggs per day.
    She eats three for breakfast every morning and bakes muffins for
    her friends every day with four.
    She sells the remainder at the farmers' market daily for $2 per
    fresh duck egg.
    How much in dollars does she make every day at the farmers'
    market?
    """
    total_eggs = 16
    breakfast_eggs = 3
    muffin_eggs = 4
    fresh_eggs = total_eggs - breakfast_eggs - muffin_eggs
    price_per_egg = 2
    total_revenue = fresh_eggs * price_per_egg
    result = total_revenue

    return result
```

Recorded execution result: 18. **Reference answer:** 18.

Figure 11: ClockDLM generation for GSM8K test example 0. The program correctly computes $(16-3-4)\times 2 = 18$.

ClockDLM — GSM8K ($S = N = 32$, 118 NFEs)

Question
A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?

Generated program

```
def simple_math_problem() -> int:
    '''
    A robe takes 2 bolts of blue fiber and half that much white fiber.
    blocking How many bolts in total does it take?
    '''
    blue_fiber = 2
    white_fiber = 0.5 * blue_fiber
    total_bolts = blue_fiber + white_fiber
    result = total_bolts
    return result
```

Recorded execution result: 3. **Reference answer:** 3.

Figure 12: ClockDLM generation for GSM8K test example 1. The program correctly computes $2+0.5\times 2 = 3$.

CNN/DailyMail RePlaid uses $T = 1024$ and ClockDLM uses $S = N = 32$, both with repetition penalty 1.2 and the remaining decoding settings in Appendix F.3. We show an excerpt of the article input, followed by the reference and both complete generated summaries.

CNN/DailyMail — Large Hadron Collider

Article excerpt
(CNN)The world's biggest and most powerful physics experiment is taking place as you read this. The Large Hadron Collider (LHC), a particle accelerator and the largest machine in the world, is ready for action following a two-year shutdown. After problems that delayed the restart in March, scientists at the European Organization for Nuclear Research (CERN) completed final tests, enabling the first beams to start circulating Sunday inside the LHC's 17 mile (27 km) ring. . . .

Reference summary
The Large Hadron Collider (LHC) begins again after a two-year shutdown .
The restart was delayed in March .

RePlaid ($T = 1024$, 1025 NFEs)
Up to 600 million particles per second at the Large Hadron Collider ready for action .
The device was join for 10 hours, traveling more than 6 billion miles (more than 10 billion km)
LHC is the largest device in 1975, in 2001, 2008, 2001 and 2001. level

ClockDLM ($S = N = 32$, 121 NFEs)
the U.S. particle accelerator is ready for action following a two-year shutdown .
A proton in the LHC makes 11,245 circuits per second .
The purpose is to recreate the conditions that existed moments after the "Big Bang"
But some still say it's not complete, but doesn't know that the universe is only matter .
The project has been developed to replicate the origin of the universe based on the particle system-like

Figure 13: Paired summaries for CNN/DailyMail test example 32. Both models receive the same article input, and their generated summaries are shown without corrections.

CNN/DailyMail — Lunar eclipse

Article excerpt
(CNN)Sky watchers in western North America are in for a treat: a nearly five-minute total lunar eclipse this morning. Here's how it's unfolding: . It started at 3:16 a.m. Pacific Daylight Time, when the moon began moving into Earth's shadow. For the next hour and 45 minutes, that shadow will move across the moon and engulf it at 4:58 a.m. Pacific Time. The total eclipse will only last four minutes and 43 seconds, and NASA says that makes it the shortest one of the century. . . .

Reference summary
The total eclipse will only last 4 minutes and 43 seconds .
People west of the Mississippi River will have the best view .
Parts of South America, India, China and Russia also will see the eclipse .

RePlaid ($T = 1024$, 1025 NFEs)
The lunar eclipse will only last four minutes and 43 seconds, when it was 3:16 a.m.
Break is third in a series of four in a row, known as a "Arrad"
The first was on April 15, 2014 .
The second was in September 2014, the next is Saturday and there will be one more, on September 28 .

ClockDLM ($S = N = 32$, 119 NFEs)
The eclipse will take for four minutes and 43 seconds .
It is the shortest one of the century, but sunrise will see the East Coast .
The first was on April 15, 2014, and the second was on September 28 .
The moon that you're up may not be good, but it's dark we can't have seen it, but it seems like it will give us a

Figure 14: Paired summaries for CNN/DailyMail test example 34. Both models receive the same article input, and their generated summaries are shown without corrections.

H ASSETS

Table 9 lists datasets used in this work. Table 10 lists tokenizers, model architectures, pretrained models, and codebases used in this work. Table 11 lists relevant software packages used in this work. For each asset, we detail its corresponding license.

Table 9: Datasets and corresponding licenses. For OpenWebText, CC0 applies to the dataset packaging; source articles retain their original rights.

Dataset	License	URL
OpenWebText (Gokaslan et al., 2019)	CC0 1.0	https://huggingface.co/datasets/Skylion007/openwebtext
TinyGSM (Liu et al., 2023)	MIT	https://huggingface.co/datasets/TinyGSM/TinyGSM
GSM8K (Cobbe et al., 2021)	MIT	https://github.com/openai/grade-school-math
CNN/DailyMail (Hermann et al., 2015; See et al., 2017)	Apache 2.0	https://huggingface.co/datasets/abisee/cnn_dailymail

Table 10: Tokenizers, architecture definitions, external code, and their corresponding licenses.

Asset	Use	License	URL
GPT-2 (Radford et al., 2019)	Tokenizer	MIT	https://huggingface.co/openai-community/gpt2
SmolLM-135M (Allal et al., 2024)	Tokenizer	Apache 2.0	https://huggingface.co/HuggingFaceTB/SmolLM-135M
Qwen3-0.6B-Base (Team, 2025)	Tokenizer, architecture	Apache 2.0	https://huggingface.co/Qwen/Qwen3-0.6B-Base
ModernBERT (Warner et al., 2025)	MAUVE Evaluation	Apache 2.0	https://huggingface.co/answerdotai/ModernBERT-large
$\mathbb{S}$-FLM (Deschenaux et al., 2026)	Baseline models, code	Apache 2.0	https://huggingface.co/jdeschena/s-flm
FMLM+ (Agarwal et al., 2026)	Baseline models, code	Apache 2.0	https://github.com/MananAg007/posterior-refinement

Table 11: Software and corresponding licenses.

Library	License
HuggingFace (Wolf et al., 2019)	Apache 2.0
Hydra (Yadan, 2019)	MIT
Language Model Evaluation Harness (Gao et al., 2021)	MIT
Matplotlib (Hunter, 2007)	Matplotlib License
MAUVE (Pillutla, 2021)	GNU General Public License, Version 3
NumPy (Harris et al., 2020)	BSD 3-Clause
OmegaConf	BSD 3-Clause
Pandas (pandas development team, 2020)	BSD 3-Clause
PyTorch (Paszke et al., 2019)	BSD 3-Clause
PyTorch Lightning (Falcon et al., 2019)	Apache 2.0
TorchMetrics	Apache 2.0