ELF: Embedded Language Flows

Keya Hu[*] **Linlu Qiu**[*] **Yiyang Lu** **Hanhong Zhao**
Tianhong Li **Yoon Kim** **Jacob Andreas** **Kaiming He**

MIT

[*]Equal contribution; order decided by a coin flip.

Code: https://github.com/lillian039/ELF

Abstract

Diffusion and flow-based models have become the *de facto* approaches for generating continuous data, *e.g.*, in domains such as images and videos. Their success has attracted growing interest in applying them to language modeling. Unlike their image-domain counterparts, today's leading diffusion language models (DLMs) primarily operate over discrete tokens. In this paper, we show that *continuous* DLMs can be made effective with minimal adaptation to the discrete domain. We propose *Embedded Language Flows (ELF)*, a class of diffusion models in continuous embedding space based on continuous-time Flow Matching. Unlike existing DLMs, ELF predominantly stays within the continuous embedding space until the final time step, where it maps to discrete tokens using a shared-weight network. This formulation makes it straightforward to adapt established techniques from image-domain diffusion models, *e.g.*, classifier-free guidance (CFG). Experiments show that ELF substantially outperforms leading discrete and continuous DLMs, achieving better generation quality with fewer sampling steps. These results suggest that ELF offers a promising path toward effective continuous DLMs.

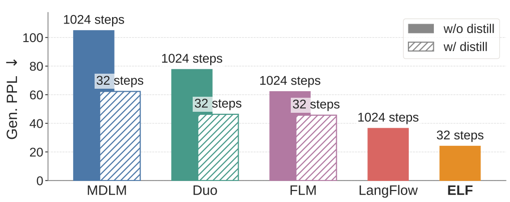

Figure 1: **ELF** achieves lower generative perplexity with fewer sampling steps than prior DLMs, without using distillation. ELF achieves this while using 10× fewer training tokens. (Model size: 105M for ELF and 170M for others; dataset: OWT. Detailed comparison in Fig. 7.)

1 Introduction

Diffusion models [63, 64, 26] and flow-based models [37, 38, 3] have become prominent paradigms for generating continuous data, demonstrating strong performance at synthesizing images, videos, and data in other continuous domains. These advances have driven growing interest in extending diffusion methods to language modeling, leading to extensive work on diffusion language models (DLMs). DLMs are commonly formulated in one of two ways: continuous or discrete. Continuous DLMs map discrete tokens into continuous representations and perform denoising in the resulting continuous space [34, 13, 19]. Discrete DLMs, in contrast, operate directly in token space and formulate a probabilistic diffusion model over discrete random variables [5, 23, 40, 56, 57]. Recent progress in DLMs has been mostly in the discrete regime, in large part due to the stronger empirical performance of discrete DLMs [33, 48, 76, 58]. But it remains an open question whether the current performance gap of continuous DLMs is due to the inherently discrete nature of language modeling or to underexplored algorithmic design choices.

In this work, we introduce Embedded Language Flows (**ELF**), a class of continuous DLMs based on Flow Matching [37, 38, 3]. ELF is continuous in two senses. First, it operates in *continuous*

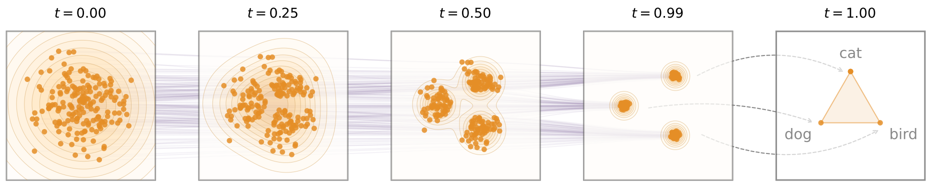

Figure 2: **Conceptual illustration of ELF.** Orange points denote data represented in continuous embedding space, and purple lines show denoising trajectories from Gaussian noise to clean embeddings. Discretization is applied only at the final time step ($t = 1$) using a shared-weight network.

embedding space by directly denoising continuous representations throughout the flowing process, with discretization considered only at the final time step. Second, it is formulated with *continuous time*, following Flow Matching [37, 38, 3], which allows us to define the velocity field via the time derivative. This formulation enables ELF to benefit from advances in Flow Matching, which is now widely used to instantiate diffusion models in image and video generation [43, 14, 6, 70].

Following Latent Diffusion Models (LDM) [54], ELF constructs the continuous embedding space by applying an encoder model to the input discrete tokens. The encoder can be pretrained, jointly trained, or frozen with random weights. *Unlike* latent diffusion, ELF does not require a separate decoder and thus introduces no additional component at inference time. This design is based on the observation that the final time step in Flow Matching can be naturally repurposed to map continuous embeddings back to discrete tokens, eliminating the need for an explicit decoder. As such, a shared-weight network is trained to perform denoising at all but the final step, and decoding (i.e. discretization) at the final step (see Fig. 2).

ELF builds on prior continuous DLMs, but aims for a minimalist design that addresses the interface between continuous and discrete spaces. In contrast to pioneering works on continuous DLMs [34, 13, 19] and many others that employ a per-step discretization loss (*e.g.*, cross-entropy), ELF performs denoising in continuous embedding space at nearly all steps, thereby offering maximal flexibility for the flow dynamics. And unlike latent diffusion methods [41, 45, 62], which typically operate in a *compressed* latent space and rely on a separate decoder, ELF directly operates in a high-dimensional latent space [32] and requires no extra decoder.

Empirically, we show that ELF outperforms leading methods on discrete DLMs and existing continuous DLMs (Fig. 1), following the evaluation protocols established in those works. ELF achieves better generation quality with fewer sampling steps than leading discrete DLMs (*e.g.*, MDLM [56] and Duo [57]) and concurrent continuous DLMs (*e.g.*, FLM [30] and LangFlow [10]). Moreover, ELF achieves this performance using $10\times$ *fewer* training tokens and *without* any distillation. We further show that ELF performs strongly on machine translation [7] and summarization [46]. Overall, these results suggest that continuous DLMs can be highly competitive while requiring only minimal treatment of discretization, offering a promising direction for diffusion-based language modeling.

2 Background & Related Work

Diffusion-/Flow-based models. Diffusion models [63, 26, 64] and flow-based models [37, 38, 2] transform noise into data through ordinary or stochastic differential equations (ODEs/SDEs). In DDPM-style formulations, generation is defined by transitions between successive states [63, 26, 47], which may be discrete or continuous. Discrete states require categorical transition distributions, as in discrete DLMs [5, 56]; continuous states are commonly modeled through score or noise prediction under Gaussian corruption [64, 26, 14]. Flow Matching extends this view to continuous time by learning the velocity field along a continuous path [37, 38, 2], where noise, data, and velocity predictions can be reparameterized into one another [14, 32]. Our method adopts Flow Matching to formulate language generation in continuous embedding space and continuous time.

Continuous diffusion language models. Continuous DLMs map discrete tokens to a continuous space to perform denoising. *Embedding-space* methods, such as Diffusion-LM [34], CDCD [13], and DiffuSeq [19], add Gaussian noise directly to token embeddings [66, 79, 21, 72, 77, 36, 74, 15]. A complementary direction studies *simplex-based* representations, including SSD-LM [22] and TESS [44, 68], as well as related manifold-based formulations [27]. Although these methods provide

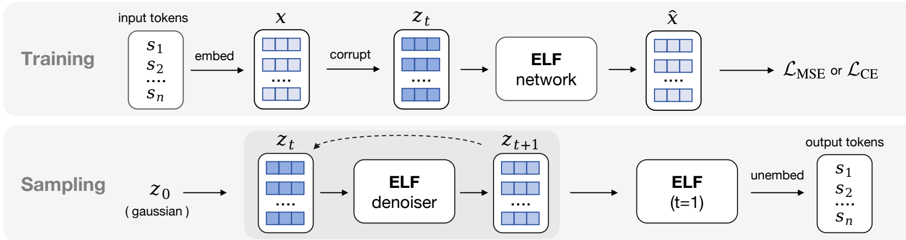

Figure 3: **During training**, discrete tokens are encoded into clean embeddings $\boldsymbol{x}$ and corrupted to $\boldsymbol{z}_t$, which ELF uses to predict $\hat{\boldsymbol{x}}$. The model is trained with either the denoising loss $\mathcal{L}_{\text{MSE}}$ or the token-wise cross-entropy loss $\mathcal{L}_{\text{CE}}$. **During inference**, ELF starts from Gaussian noise $\boldsymbol{z}_0$ and iteratively denoises embeddings from $\boldsymbol{z}_t$ to $\boldsymbol{z}_{t+1}$. Only at the final step does ELF switch to decoding mode and project the final embeddings back to discrete tokens through an unembedding layer.

continuous relaxations of discrete tokens, their trajectories often remain tied to the discrete token space through mechanisms such as rounding losses, simplex constraints, and token-level cross-entropy objectives. In contrast, ELF denoises entirely in continuous embedding space without per-step token-level supervision and discretizes only at the final step.

Another line applies *latent diffusion* to frozen encoder representations, represented by LD4LG [41] and follow-up work [81, 59, 42, 45, 62]. Like many diffusion methods described above, these approaches typically follow DDPM-style or score-based formulations with DDPM noise schedules [26, 47], and additionally rely on a separately trained decoder to recover tokens. In contrast, ELF uses a continuous-time Flow Matching formulation with a linear (rectified-flow) interpolant [37, 38, 2], and does not require a separate decoder. This brings flow-based training and sampling into language diffusion, allowing ELF to benefit from recent advances in Flow Matching.

Several concurrent works also revisit continuous flow-based language modeling. DFM [51], CFM [55], FLM/FMLM [30], and LangFlow [10] all incorporate token-level cross-entropy supervision along the flow trajectory, though they differ in the continuous state space, including simplex space, one-hot token encodings, and embedding space. Some of these methods further introduce distillation for few-step generation, such as distilled DFM/CFM and FMLM. In contrast, ELF keeps the denoising trajectory entirely in an unrestricted continuous embedding space, applying token-level supervision only at the final decoding step. A more comprehensive survey is provided in Appendix A.

Discrete diffusion language models. Due to the discrete nature of language, another line of work applies diffusion directly in token space. D3PMs [5] define general discrete corruption processes, including absorbing and uniform transitions. Masked diffusion models, such as MDLMs [56], use a special `[MASK]` absorbing state and generate samples through iterative unmasking [23, 48, 76]. Subsequent work improves sampling and efficiency through remasking, adaptive inference [71, 73], and semi-autoregressive block diffusion, including E2D2 [4]. Uniform-state diffusion models, such as Duo [57], instead diffuse tokens toward a uniform categorical distribution, enabling repeated token revision during inference [57, 12, 58]. Recent studies further scale discrete DLMs and extend them to code and multimodal generation [20, 65, 75, 78, 31]. Overall, discrete diffusion models currently remain the dominant paradigm in diffusion-based language modeling [33].

3 Embedded Language Flows

In this section, we present our flow-based formulation for language modeling (Fig. 3). Our method leverages the iterative nature of flow models to perform denoising primarily in continuous embedding space, converting clean embeddings back to discrete tokens only at the final step. Following prior work [56, 57, 30, 10], we describe our method in the simpler setting of unconditional generation. The framework can be extended to conditional generation, as discussed in Sec. 3.3.

3.1 The ELF Framework

From discrete tokens to continuous embeddings. To apply continuous diffusion to language, we first map discrete tokens to continuous representations. Given a sentence, we tokenize it into a sequence of tokens $\boldsymbol{s} = [s_1, \ldots, s_L] \in V^L$, where each s_i is drawn from the vocabulary V

and L denotes the sequence length. We then map the discrete token sequence into a continuous embedding space. The choice of the embedding method is flexible. By default, we use a pretrained T5 encoder [53] for bidirectional contextual embeddings. We also explore other jointly trained and randomized embeddings (see Sec. 4.1). The encoder is only used during training, which does not incur additional modules at inference.

Flow Matching on continuous embeddings. After obtaining continuous language representations, we formulate the denoising process in the resulting embedding space using Flow Matching [37, 38, 3]. Flow Matching defines a continuous flow path from noise to data in this space. Let $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$ denote the embedding distribution and $\boldsymbol{\epsilon} \sim p_{\text{noise}}(\boldsymbol{\epsilon})$ denote the noise distribution (*e.g.*, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$). The noisy latent variable is defined by linear interpolation ("rectified flows"): $\boldsymbol{z}_t = t\boldsymbol{x} + (1-t)\boldsymbol{\epsilon}$, where $t \in [0,1]$, and $\boldsymbol{z}_0 \sim p_{\text{noise}}$ and $\boldsymbol{z}_1 \sim p_{\text{data}}$. In continuous time, the flow velocity $\boldsymbol{v}$ is defined as the time derivative of $\boldsymbol{z}$, that is, $\boldsymbol{v} = d\boldsymbol{z}/dt = \boldsymbol{x} - \boldsymbol{\epsilon}$.

While standard Flow Matching directly parameterizes $\boldsymbol{v}$ via a neural network, ELF follows recent advances on image generation and instead parameterizes $\boldsymbol{x}$ [32] (**$\boldsymbol{x}$-prediction**). Specifically, let $\boldsymbol{x}_\theta = \texttt{net}_\theta(\boldsymbol{z}_t, t)$ denote the network's immediate output. We train the model by minimizing the mean squared error (MSE) between the predicted velocity and the target velocity:

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}_{t,\boldsymbol{x},\boldsymbol{\epsilon}} \|\boldsymbol{v}_\theta(\boldsymbol{z}_t, t) - \boldsymbol{v}\|^2 = \mathbb{E}_{t,\boldsymbol{x},\boldsymbol{\epsilon}} \frac{1}{(1-t)^2} \|\boldsymbol{x}_\theta(\boldsymbol{z}_t, t) - \boldsymbol{x}\|^2, \tag{1}$$

where we leverage the relation $\boldsymbol{v}(\boldsymbol{z}_t, t) = (\boldsymbol{x} - \boldsymbol{z}_t)/(1-t)$ [32].

The $\boldsymbol{x}$-prediction parameterization is important for ELF. First, it enables Flow Matching to perform effectively on high-dimensional representations (*e.g.*, 768-d per-token embeddings), consistent with observations in [32] (see Appendix C.1 for ELF's ablations on prediction targets). Second, predicting clean embeddings (*i.e.*, $\boldsymbol{x}$) aligns naturally with the objective of predicting clean discrete tokens at the final step (discussed next), whereas the standard $\boldsymbol{v}$-prediction in Flow Matching does not. Although $\boldsymbol{v}$ can be predicted by a network and transformed into $\boldsymbol{x}$, the weight sharing that ties the denoising (MSE loss) and decoding (cross-entropy loss) objectives is compromised. Empirically, we observe that $\boldsymbol{v}$-prediction works poorly when weights are shared with the final discretization step.

Back to discrete tokens. As the generation output consists of discrete tokens, we convert the clean embeddings back into tokens at the final time step (*i.e.*, at $t = 1$). By considering the final time step of ELF naturally as continuous-to-discrete decoding, our method does not require a separate decoder (or equivalently, it can be thought of as a decoder sharing weights with the denoiser).

The network input at this time step should be $\boldsymbol{z}_t$ in the limit $t \to 1$. But because $\boldsymbol{z}_t \to \boldsymbol{x}$ as $t \to 1$, we introduce a token-level corruption process at this final step to create a nontrivial training input, denoted as $\tilde{\boldsymbol{z}}$ (detailed in Appendix B.1). The same network $\texttt{net}_\theta$ maps $\tilde{\boldsymbol{z}}$ to a clean embedding $\boldsymbol{x}_\theta(\tilde{\boldsymbol{z}})$, which is subsequently projected by a learnable "unembedding" matrix W to obtain logits. We minimize a per-token cross-entropy (CE) loss against the ground-truth token $\boldsymbol{s}$:

$$\mathcal{L}_{\text{CE}} = \mathbb{E}_{\tilde{\boldsymbol{z}}} \left[\text{CrossEnt}(W\boldsymbol{x}_\theta(\tilde{\boldsymbol{z}}), \boldsymbol{s})\right], \tag{2}$$

The network $\boldsymbol{x}_\theta$ shares weights with that in Eq. (1) and is conditioned on a binary "mode" token (denoise or decode) in addition to the time condition $t = 1$. At inference time, we evaluate $W\boldsymbol{x}_\theta(\boldsymbol{z}_t)$ only at the final step $t = 1$, and apply argmax to obtain a discrete token.

3.2 Pseudocode

The core concepts of ELF are summarized in Alg. 1 and Alg. 2 (detailed in Appendix Fig. 9).

Training. As in standard Flow Matching, ELF employs a single network $\texttt{net}_\theta$ to model all time steps, conditioned on t. This includes the final time step $t = 1$, which uses different pre-processing (corruption) and post-processing (loss computation). For clarity, we illustrate this distinction using an explicit "`if`" branch in Alg. 1. In practice, samples from both branches are processed within a *single* batch, and masking is used to selectively apply the appropriate corruption and unembedding operations as well as the corresponding loss terms. The network is further conditioned on a binary "mode" token that indicates whether the operation is "denoise" or "decode".

Inference. During inference, ELF iteratively transforms noisy samples into clean embeddings. Starting from $\boldsymbol{z}_0 \sim \mathcal{N}(0, \mathbf{I})$, ELF solves the ODE: $d\boldsymbol{z}_t/dt = \boldsymbol{v}_\theta(\boldsymbol{z}_t, t)$, which is approximated with

Algorithm 1 ELF: training.

Two-branch computation is batched, adding no extra training cost.

```
# net(z, t, mode): ELF network
# s: a sequence of discrete tokens

x = encode(s)
if uniform(0, 1) < threshold:
    # denoising branch
    t = sample_t()
    e = randn_like(x)
    z = t * x + (1 - t) * e
    v = x - e
    x_pred = net(z, t, mode="denoise")
    v_pred = (x_pred - z) / (1 - t)
    loss = mse_loss(v_pred, v)
else:
    # decoding branch (t = 1)
    z = corrupt(x)
    x_pred = net(z, t=1, mode="decode")
    s_pred = unembed(x_pred)
    loss = ce_loss(s_pred, s)
```

Algorithm 2 ELF: inference.

We show ODE for simplicity. SDE sampler is also applicable.

```
# shape: shape of embedded sequences
# ts: sampling time schedule, from 0 to 1

z = randn(shape)
for i in range(len(ts) - 1):
    t = ts[i]
    dt = ts[i + 1] - ts[i]
    x_pred = net(z, t, mode="denoise")

    # convert x prediction to velocity
    v = (x_pred - z) / (1 - t)
    z = z + dt * v

# final step
h = net(z, t=1, mode="decode")

# unembedding
token_logits = unembed(h)
tokens = argmax(token_logits)
```

a numerical (*e.g.*, Euler) solver. At the final time step $t = 1$, we apply the network under the "decode" mode and perform unembedding and discretization.

Besides the ODE formulation, our method also supports an SDE-inspired sampler. The underlying SDE associated with Flow Matching can be derived following [43], where the dynamics can be interpreted as injecting infinitesimal noise at each step. In practice, we adopt a simpler approximation to emulate this behavior: we inject small noise at each step while correspondingly shifting the time variable t toward the noise regime (detailed in Appendix, Alg. 6). For brevity, we refer to the resulting SDE-inspired sampler as the "SDE" variant, while noting that it primarily captures the per-step stochastic behavior. We experimentally compare the ODE formulation with this SDE variant.

3.3 Conditioning and Guidance

Controlling model generation is an important aspect of generative modeling. In image diffusion models, classifier-free guidance (CFG) [25] has been established as a highly effective technique for steering the generated output.[1] CFG also enables a trade-off between generation quality and diversity. Because CFG was originally formulated for continuous quantities (*e.g.*, score functions or velocity fields), it is naturally applicable to ELF. This stands in contrast to discrete counterparts, where CFG remains largely unexplored and has been shown less effective [30, 51].

In the absence of class labels, we employ *self-conditioning* [9] to construct the conditioning signals required for CFG. Given that self-conditioning is already a standard component in DLMs [79, 13, 66, 41, 44, 59, 60], incorporating CFG introduces only marginal computational overhead. In what follows, we first describe the self-conditioning used in ELF and then introduce CFG.

Self-conditioning. In a standard Flow Matching model (*i.e.*, without self-conditioning), a forward pass at a given time step yields a single prediction. We denote this prediction by $\hat{\boldsymbol{x}}'$ in our case, indicating that it corresponds to a prediction of the clean embedding $\boldsymbol{x}$. During training, self-conditioning [9] performs a second forward pass, conditioned on $\hat{\boldsymbol{x}}'$, which serves as an intermediate prediction. The output of the second pass, denoted as $\hat{\boldsymbol{x}}$, can be written as $\hat{\boldsymbol{x}} = \texttt{net}_\theta(\boldsymbol{z}_t \mid \hat{\boldsymbol{x}}', t)$. This is implemented by concatenating $[\boldsymbol{z}_t, \hat{\boldsymbol{x}}']$ as the network input [9]. During training, the model is conditioned on $\hat{\boldsymbol{x}}'$ with probability 50%, and uses a null condition $\mathbf{0}$ otherwise (see Appendix, Fig. 9 for details). During inference, the model conditions on the prediction from the previous time step, thus introducing no extra forward passes for inference.

The intermediate prediction $\hat{\boldsymbol{x}}'$ serves as a condition for the network. As such, it can be treated as the conditioning signal $\boldsymbol{c}$ in the application of CFG, introduced next.

[1] CFG was historically introduced for *class*-conditional generation. However, the notion of a condition can be generalized to other inputs, *e.g.*, a text prompt. We use CFG in this broader sense, as our setting does not involve class labels.

CFG with self-conditioning. CFG [25] combines the unconditional and conditional predictions through a linear extrapolation. Formally, given a conditioning signal $\boldsymbol{c}$, CFG in Flow Matching defines a velocity field as $\boldsymbol{v}_{\text{cfg}}(\boldsymbol{z}_t \mid \boldsymbol{c}) = \omega \boldsymbol{v}(\boldsymbol{z}_t \mid \boldsymbol{c}) + (1-\omega)\boldsymbol{v}(\boldsymbol{z}_t \mid \varnothing)$, where $\varnothing$ denotes the unconditional counterpart and ω is the guidance scale. As discussed, our conditioning signal $\boldsymbol{c}$ is obtained from self-conditioning. In its original form [25], CFG is applied at inference time, requiring two forward passes per step.

To avoid inference-time overhead, we adopt *training-time* CFG techniques [8, 69, 16, 17] previously developed for image generation. These methods use a single network pass to model $\boldsymbol{v}_{\text{cfg}}$ instead of $\boldsymbol{v}$ (in our case, $\boldsymbol{x}_{\text{cfg}}$ instead of $\boldsymbol{x}$). Because ELF is formulated similarly to its image-generation counterpart, adapting it to training-time CFG is straightforward, further illustrating the advantages of our continuous-based formulation. The implementation details, following the form in [16, 17], are in Appendix (Alg. 3, 4, & 5).

Extension to conditional generation. Thus far, we have presented our method in the setting of unconditional generation, as in prior work [56, 57, 30, 10]. Our method can be naturally extended to conditional generation, in which outputs are conditioned on an input sequence (*e.g.*, a prompt). In this setting, we prepend the clean embeddings of the conditioning sequence to the model input and preserve them without corruption during both training and inference. The model can then condition on them through self-attention.

CFG remains applicable in the conditional setting. The conditioning $\boldsymbol{c}$ now consists of both the self-conditioning and the prefix clean embeddings; the unconditional counterpart is obtained by zeroing out $\boldsymbol{c}$. Analogous to text-to-image generation [14], CFG is effective in controlling generation quality in our scenario, which can be viewed as "text-to-text" generation.

4 Experiments

Dataset and evaluation. For unconditional generation, we follow the experimental design used in past work [56, 57, 30, 10]. We train on the OpenWebText (OWT) dataset [18], which has around 9B tokens, and pack sequences to length $L = 1024$. For evaluation, we generate 1,000 samples and report generative perplexity (Gen. PPL), *i.e.*, the perplexity of generated samples under a pretrained GPT-2 Large model [52]; together with average unigram entropy as a measure of sample diversity.[2]

For conditional generation, we consider machine translation and summarization. For machine translation, we use the WMT14 German-to-English (De-En) dataset [7] with sequence length $L = 128$ (condition length 64, target length 64; 144M total target tokens), and evaluate using BLEU [49]. For summarization, we use the XSum dataset [46] with sequence length $L = 1088$ (condition length 1024, target length 64; 6M total target tokens), and report ROUGE-1 (R1), ROUGE-2 (R2), and ROUGE-L (R-L) [35]. We treat both as sequence-to-sequence tasks and do not use sequence packing for conditional generation.

Model. We use contextual embeddings from a frozen pretrained T5-small encoder [53] (35M) with embedding dimension 512. We use a bottleneck design that linearly projects embeddings into a lower-dimensional space of size 128, and then projects them back to the hidden size of the model [32]. We consider three model sizes: ELF-B (105M), ELF-M (342M), and ELF-L (652M), and use ELF-B as the default for ablations. Detailed configurations are shown in Appendix Tab. 3.

Training and inference. We train our model using the Muon optimizer [28] with a learning rate of 0.002 and a batch size of 512. The model is trained for 5 epochs on OWT (around 95K steps), and for 100 epochs on WMT14 and XSum (around 880K and 40K steps, respectively). Depending on the selected model mode, the network is trained with either the MSE loss in Eq. 1 (80%) or the CE loss in Eq. 2 (20%). During inference, we use the ODE or SDE sampler to generate samples.

4.1 Ablations

We begin by ablating several key design choices of our model on the simpler setting of unconditional generation on OWT, using the default ELF-B model and a 64-step ODE Euler sampler unless otherwise specified. More ablation studies are shown in Appendix C.

[2]We do not use validation perplexity, since likelihood evaluation for flow-based models can require additional likelihood-specific training [1].

Classifier-free guidance (CFG). Our flow-based continuous formulation is naturally compatible with CFG, a highly effective technique in standard diffusion models. Therefore, we first study the effect of the CFG scale. As shown in Fig. 4, increasing the CFG scale lowers generative perplexity but also reduces entropy, reflecting a quality–diversity trade-off. The preferred direction is toward the lower-right region of the plot, corresponding to lower generative perplexity and higher entropy. For most of the remaining ablations, we evaluate this quality–diversity trade-off by sweeping the CFG scale. Each point on the curve is computed from 1,000 generated samples at a specific CFG scale.

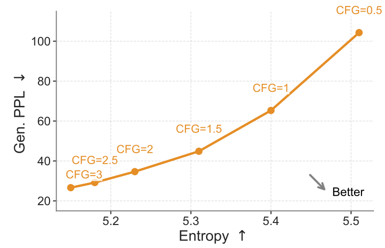

Figure 4: **Ablations on guidance.** We evaluate the generative perplexity–entropy trade-off across CFG scales: increasing the scale lowers generative perplexity but reduces entropy.

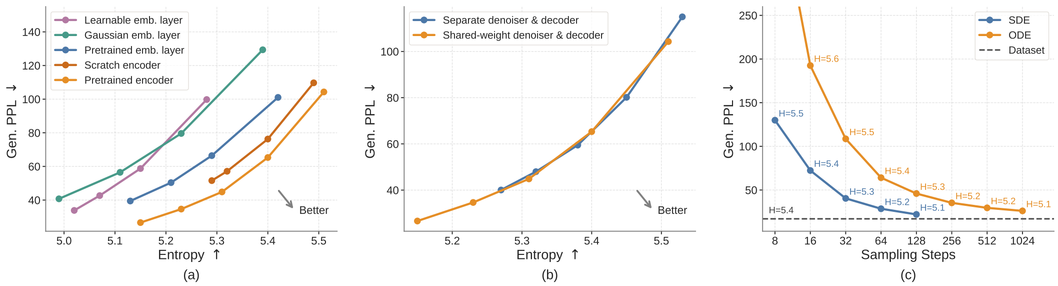

Figure 5: **Ablations on key design choices.** (a) Embedding choices: we compare contextual *vs.* non-contextual embeddings, as well as frozen *vs.* learnable embeddings; pretrained contextual embeddings achieve the best trade-off. (b) Decoding strategies: We compare a shared-weight denoiser-decoder with a two-stage, separately trained decoder. Both strategies achieve similar trade-offs, but the shared-weight variant extends further toward the regime of low generative perplexity. (c) Samplers: we compare ODE and SDE-inspired samplers across different sampling steps; SDE-inspired sampler consistently achieves lower generative perplexity in fewer steps.

Embedding choices. Since ELF operates in a continuous embedding space, we next study how the choice of embeddings affects performance. We ablate the continuous embeddings along two axes: whether the embeddings are contextual (*i.e.*, from an encoder) or non-contextual (*i.e.*, from a single embedding layer), and whether they are fixed or learnable. For contextual embeddings, we evaluate those from an off-the-shelf T5 encoder [53] and embeddings from an encoder trained from scratch on OWT using the original T5 objective. For non-contextual embeddings, we consider token embeddings from the pretrained T5 model, frozen Gaussian embeddings, and learnable embeddings. See Appendix D.3 for detailed setup. We show the results in Fig. 5a. Contextual embeddings achieve a better generative perplexity–entropy trade-off. Embeddings from an encoder trained from scratch on OWT perform well, but slightly lag behind those from a pretrained encoder. Among the non-contextual variants, pretrained token embeddings outperform frozen Gaussian embeddings. Learnable embeddings perform the worst, likely due to the difficulty of jointly optimizing the embeddings and the denoiser. Overall, these results suggest that *pretrained contextual embeddings* are favorable representations of language for ELF.

Decoding strategies. Since we use contextual embeddings as our continuous representations, we need to decode them back into discrete tokens. We use a shared-weight network, with training interleaving $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{CE}}$. Alternatively, we explore a two-stage strategy. In the first stage, we train a decoder from scratch with a frozen pretrained T5 encoder to reconstruct tokens from masked and noisy embeddings using $\mathcal{L}_{\text{CE}}$. In the second stage, we freeze both the encoder and decoder, and train a separate denoiser using $\mathcal{L}_{\text{MSE}}$ (see Appendix D.3 for details). As shown in Fig. 5b, both strategies achieve a similar trade-off, but the shared-weight variant extends further toward the regime of low generative perplexity , while also simplifying the pipeline by avoiding an extra training stage.

Samplers. Since ELF is formulated in continuous time and continuous space, it naturally supports both deterministic ODE sampling and stochastic SDE-like sampling; see Appendix Alg. 6 for details. We compare ODE and SDE samplers across different sampling budgets with a self-conditioning CFG scale of 1. As shown in Fig. 5c, SDE sampling achieves substantially lower generative perplexity than

ODE sampling in the few-step regime. These results suggest that introducing stochasticity during sampling can effectively reduce error accumulation and provide a better quality–efficiency trade-off.

Model scales. We study the scaling behavior of ELF across three model sizes: **ELF-B** (105M), **ELF-M** (342M), and **ELF-L** (652M) (detailed in Appendix Tab. 3). We evaluate each model using both ODE and SDE sampling. As shown in Fig. 6, scaling consistently improves the generative perplexity–entropy frontier. In particular, at matched entropy, larger models achieve lower generative perplexity, indicating higher sample quality with comparable diversity. Conversely, at similar generative perplexity, larger models maintain higher entropy. The effect of the sampler is consistent across model sizes: SDE sampling improves over ODE sampling by pushing the frontier in a more optimal direction. These results suggest that ELF scales effectively, demonstrating the potential of model scaling. See Appendix Tab. 7 for the detailed numbers.

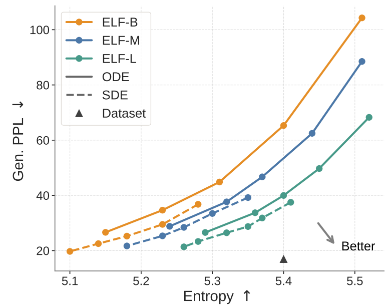

Figure 6: **Scaling of ELF models.** We compare ELF-B, ELF-M, and ELF-L. Scaling model size consistently improves the Gen. PPL–entropy frontier.

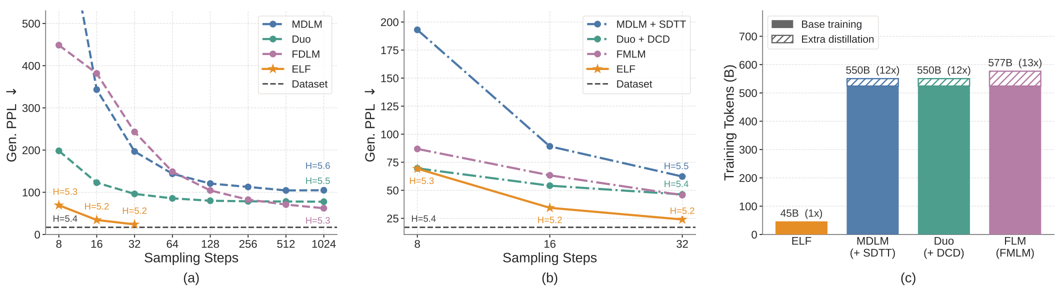

Figure 7: **System-level comparison.** ELF-B outperforms both discrete and continuous DLMs trained under similar settings (a), rivals distilled variants of other baselines that require additional rounds of training (b), and uses substantially fewer training tokens (c).

4.2 System-Level Comparison on Unconditional Generation

We first compare ELF-B against both discrete DLMs, including MDLM [56] and Duo [57], and continuous DLMs, including FLM [30] and LangFlow [10], under a comparable setting. All models are trained on the OWT dataset. ELF has 105M parameters, while the compared baselines have around 170M parameters. For ELF, we use our best configuration: SDE sampling with self-conditioning CFG scale of 3 (see Appendix D.2 for details). We show results in Fig. 7a. ELF achieves a generative perplexity of 24 using only 32 sampling steps, requiring substantially less inference-time compute than prior methods. ELF remains strong even compared with distilled models, which require extra training to distill a student model for few-step generation. As shown in Fig. 7b, in the few-step regime, ELF outperforms distilled models, including MDLM+SDTT [56, 11], Duo+DCD [57], and FMLM [30], even without any additional distillation.

ELF is also substantially more data-efficient in terms of estimated training tokens, as shown in Fig. 7c. While prior DLMs typically use over 500B tokens, ELF uses only 45B.[3] Together, these results show that, when combined with proper sampling and guidance, ELF achieves strong system-level performance. It not only improves inference efficiency, but also achieves strong performance with a much smaller training budget, demonstrating the potential of our flow-based language model. See Fig. 8 for qualitative examples of ELF-B's generations.

4.3 System-Level Comparison on Conditional Generation

We compare ELF-B with autoregressive and diffusion-based baselines at a similar model scale. These include discrete DLMs (MDLM [56], Duo [57], and E2D2 [4]) and continuous DLMs (SeqDif-

[3]A per-method breakdown of training token counts is provided in Appendix Tab. 5. We also experimented with training on more tokens, but did not observe further performance improvement.

Model	Size	De-En† BLEU ↑	XSum‡ ROUGE-1 ↑	ROUGE-2 ↑	ROUGE-L ↑
AR	99M	25.2	30.5 $\pm$ 0.13	10.2 $\pm$ 0.11	24.4 $\pm$ 0.12
MDLM [56]	99M	18.4	33.4 $\pm$ 0.11	11.6 $\pm$ 0.10	25.8 $\pm$ 0.10
Duo [57]	170M (+35M)	21.3‡	31.4 $\pm$ 0.12	10.1 $\pm$ 0.10	25.0 $\pm$ 0.12
E2D2 [4]	99M	24.8	28.4 $\pm$ 0.11	8.3 $\pm$ 0.09	22.0 $\pm$ 0.10
SeqDiffuSeq [79]	-	21.3	19.3†	1.7†	14.1†
CDCD [13]	-	24.9	-	-	-
Ours	105M (+35M)	**26.4**	**36.0** $\pm$ 0.13	**12.2** $\pm$ 0.11	**27.8** $\pm$ 0.12

Table 1: **Results on machine translation and summarization.** We evaluate ELF-B on WMT14 German-to-English (De-En) translation and XSum summarization, comparing against baselines of similar parameter scale. † denotes results taken directly from prior work and is the default source for De-En, while ‡ denotes results we reproduced using public codebases and is the default source for XSum. For XSum, we additionally report the standard error across evaluation examples when available. ELF achieves the best performance on both settings.

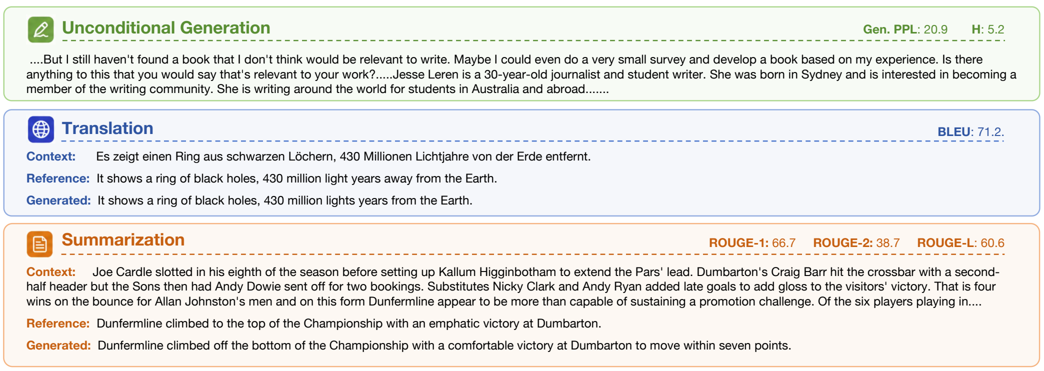

Figure 8: **Qualitative examples** of text generated by ELF-B. We show an unconditional sample, a German-to-English translation example, and a summarization example, along with their automatic evaluation metrics. Some text is omitted due to space limits; see Appendix E for more examples.

fuSeq [79] and CDCD [13]). Some results are taken from the literature and others are reproduced from public codebases. See Appendix Tab. 8 for a summary. We use the best sampling configuration selected on the validation set: a 64-step ODE sampler with the self-conditioning CFG scale set to 1 and the input-condition CFG scale set to 2.

We show the results in Tab. 1. On WMT14 De–En, ELF-B achieves a BLEU score of 26.4, outperforming all compared baselines. On XSum, ELF-B also outperforms all compared baselines across all ROUGE metrics. These results demonstrate the effectiveness of ELF on conditional generation tasks. Qualitative examples in Fig. 8 show that ELF-B generally follows the input context and produces outputs that semantically align with the ground-truth references.

5 Conclusion

We introduced **Embedded Language Flows** (ELF), a continuous diffusion language model that formulates language generation in continuous embedding space using continuous-time Flow Matching. In contrast to prior DLMs, ELF keeps the denoising trajectory continuous and applies discretization only at the final step, enabling straightforward adaptation of techniques from continuous diffusion models. Empirically, compared with leading discrete DLMs and existing continuous DLMs, ELF achieves a strong quality–efficiency trade-off across language generation tasks, attaining lower generative perplexity with fewer sampling steps and fewer training tokens. These results suggest that continuous DLMs remain a promising direction for diffusion-based language modeling.

Acknowledgments and Disclosure of Funding

We thank Mingyang Deng, Belinda Li, Itamar Pres, and Laura Ruis, for their helpful feedback and insightful discussions. We thank Google TPU Research Cloud (TRC) for granting us access to TPUs.

References

[1] Xinyue Ai, Yutong He, Albert Gu, Ruslan Salakhutdinov, J Zico Kolter, Nicholas Matthew Boffi, and Max Simchowitz. Joint distillation for fast likelihood evaluation and sampling in flow-based models. In *ICLR*, 2026. 6

[2] Michael Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *JMLR*, 2025. 2, 3, 15

[3] Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *ICLR*, 2023. 1, 2, 4

[4] Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. In *NeurIPS*, 2025. 3, 8, 9, 27

[5] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, 2021. 1, 2, 3

[6] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. FLUX.1 Kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025. 2

[7] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ales Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *ACL Workshop on Statistical Machine Translation*, 2014. 2, 6

[8] Huayu Chen, Kai Jiang, Kaiwen Zheng, Jianfei Chen, Hang Su, and Jun Zhu. Visual generation without guidance. In *ICML*, 2025. 6, 18

[9] Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *ICLR*, 2023. 5, 18

[10] 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. 2, 3, 6, 8, 15, 25

[11] Justin Deschenaux and Caglar Gulcehre. Beyond autoregression: Fast LLMs via self-distillation through time. In *ICLR*, 2025. 8

[12] Justin Deschenaux, Caglar Gulcehre, and Subham Sekhar Sahoo. The diffusion duality, chapter ii: ψ-samplers and efficient curriculum. In *ICLR*, 2026. 3

[13] Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022. 1, 2, 5, 9, 15, 27

[14] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow Transformers for high-resolution image synthesis. In *ICML*, 2024. 2, 6

[15] Zhujin Gao, Junliang Guo, Xu Tan, Yongxin Zhu, Fang Zhang, Jiang Bian, and Linli Xu. Empowering diffusion models on the embedding space for text generation. In *NAACL*, 2024. 2, 15

[16] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *NeurIPS*, 2025. 6, 18

[17] Zhengyang Geng, Yiyang Lu, Zongze Wu, Eli Shechtman, J Zico Kolter, and Kaiming He. Improved mean flows: On the challenges of fastforward generative models. *arXiv preprint arXiv:2512.02012*, 2025. 6, 18

[18] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019. 6, 25

[19] Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. In *ICLR*, 2023. 1, 2, 15

[20] Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. In *ICLR*, 2026. 3

[21] Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. In *NeurIPS*, 2023. 2, 15

[22] Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. SSD-LM: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *ACL*, 2023. 2, 15

[23] Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuan-Jing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. In *ACL*, 2023. 1, 3

[24] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for Transformers. In *Findings of EMNLP*, 2020. 24

[25] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS Workshops*, 2021. 5, 6

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 1, 2, 3, 15, 16

[27] Jaehyeong Jo and Sung Ju Hwang. Continuous diffusion model for language modeling. In *NeurIPS*, 2025. 2, 15

[28] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cecista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. Technical report, Keller Jordan blog, 2024. 6, 23

[29] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *NeurIPS*, 2022. 23

[30] 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. 2, 3, 5, 6, 8, 15, 25

[31] Lijiang Li, Zuwei Long, Yunhang Shen, Heting Gao, Haoyu Cao, Xing Sun, Caifeng Shan, Ran He, and Chaoyou Fu. Omni-diffusion: Unified multimodal understanding and generation with masked discrete diffusion. *arXiv preprint arXiv:2603.06577*, 2026. 3

[32] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. *arXiv preprint arXiv:2511.13720*, 2025. 2, 4, 6, 20, 21, 22

[33] Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*, 2025. 1, 3

[34] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-LM improves controllable text generation. In *NeurIPS*, 2022. 1, 2, 15

[35] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *ACL Workshop on Text Summarization Branches Out*, 2004. 6

[36] Zhenghao Lin, Yeyun Gong, Yelong Shen, Tong Wu, Zhihao Fan, Chen Lin, Nan Duan, and Weizhu Chen. Text generation with diffusion language models: A pre-training approach with continuous paragraph denoise. In *ICML*, 2023. 2, 15

[37] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023. 1, 2, 3, 4, 15

[38] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023. 1, 2, 3, 4, 15

[39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 23

[40] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *ICML*, 2024. 1

[41] Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q Weinberger. Latent diffusion for language generation. In *NeurIPS*, 2023. 2, 3, 5, 15, 27

[42] Justin Lovelace, Varsha Kishore, Yiwei Chen, and Kilian Q Weinberger. Diffusion guided language modeling. In *Findings of ACL*, 2024. 3, 15

[43] Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. SiT: Exploring flow and diffusion-based generative models with scalable interpolant Transformers. In *ECCV*, 2024. 2, 5, 19

[44] Rabeeh Karimi Mahabadi, Hamish Ivison, Jaesung Tae, James Henderson, Iz Beltagy, Matthew E Peters, and Arman Cohan. Tess: Text-to-text self-conditioned simplex diffusion. In *EACL*, 2024. 2, 5, 15

[45] Viacheslav Meshchaninov, Egor Chimbulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry Vetrov. Cosmos: Compressed and smooth latent space for text diffusion modeling. In *NeurIPS*, 2025. 2, 3, 15

[46] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *EMNLP*, 2018. 2, 6

[47] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021. 2, 3, 16

[48] 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. In *NeurIPS*, 2025. 1, 3

[49] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL*, 2002. 6

[50] William Peebles and Saining Xie. Scalable diffusion models with Transformers. In *ICCV*, 2023. 18, 24

[51] Peter Potaptchik, Jason Yim, Adhi Saravanan, Peter Holderrieth, Eric Vanden-Eijnden, and Michael S Albergo. Discrete flow maps. *arXiv preprint arXiv:2604.09784*, 2026. 3, 5, 15

[52] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. 6

[53] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020. 4, 6, 7, 25

[54] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 2

[55] Daan Roos, Oscar Davis, Floor Eijkelboom, Michael Bronstein, Max Welling, İsmail İlkan Ceylan, Luca Ambrogioni, and Jan-Willem van de Meent. Categorical flow maps. *arXiv preprint arXiv:2602.12233*, 2026. 3, 15

[56] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *NeurIPS*, 2024. 1, 2, 3, 6, 8, 9, 25

[57] Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. In *ICML*, 2025. 1, 2, 3, 6, 8, 9, 25, 27

[58] Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026. 1, 3

[59] Alexander Shabalin, Viacheslav Meshchaninov, Egor Chimbulatov, Vladislav Lapikov, Roman Kim, Grigory Bartosh, Dmitry Molchanov, Sergey Markov, and Dmitry Vetrov. TEncDM: Understanding the properties of the diffusion model in the space of language model encodings. In *AAAI*, 2025. 3, 5, 15

[60] Alexander Shabalin, Simon Elistratov, Viacheslav Meshchaninov, Ildus Sadrtdinov, and Dmitry Vetrov. Why gaussian diffusion models fail on discrete data? *arXiv preprint arXiv:2604.02028*, 2026. 5

[61] Noam Shazeer. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020. 24

[62] Junzhe Shen, Jieru Zhao, Ziwei He, and Zhouhan Lin. Codar: Continuous diffusion language models are more powerful than you think. *arXiv preprint arXiv:2603.02547*, 2026. 2, 3, 15

[63] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015. 1, 2

[64] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021. 1, 2, 15

[65] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025. 3

[66] Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, and Rémi Leblond. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022. 2, 5, 15

[67] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 24

[68] Jaesung Tae, Hamish Ivison, Sachin Kumar, and Arman Cohan. Tess 2: A large-scale generalist diffusion language model. In *ACL*, 2025. 2, 15

[69] Zhicong Tang, Jianmin Bao, Dong Chen, and Baining Guo. Diffusion models without classifier-free guidance. *arXiv preprint arXiv:2502.12154*, 2025. 6, 18

[70] Wan Team, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. 2

[71] Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *NeurIPS*, 2025. 3

[72] Renzhi Wang, Jing Li, and Piji Li. InfoDiffusion: Information entropy aware diffusion process for non-autoregressive text generation. In *Findings of EMNLP*, 2023. 2, 15

[73] 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. In *ICLR*, 2026. 3

[74] Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, Jian Jiao, Juntao Li, Jian Guo, Nan Duan, and Weizhu Chen. AR-Diffusion: Auto-regressive diffusion model for text generation. In *NeurIPS*, 2023. 2, 15

[75] Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. In *NeurIPS*, 2025. 3

[76] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025. 1, 3

[77] Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Mingxuan Wang. DINOISER: Diffused conditional sequence learning by manipulating noises. *Transactions of the Association for Computational Linguistics*, 2024. 2, 15

[78] Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. Llada-v: Large language diffusion models with visual instruction tuning. *arXiv preprint arXiv:2505.16933*, 2025. 3

[79] Hongyi Yuan, Zheng Yuan, Chuanqi Tan, Fei Huang, and Songfang Huang. Seqdiffuseq: Text diffusion with encoder-decoder transformers. In *NAACL*, 2024. 2, 5, 9, 15

[80] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019. 24

[81] Yizhe Zhang, Jiatao Gu, Zhuofeng Wu, Shuangfei Zhai, Josh Susskind, and Navdeep Jaitly. PLANNER: Generating diversified paragraphs via latent language diffusion model. In *NeurIPS*, 2023. 3, 15

Method	Process[†]	State[‡]	Train per-step discr.	Infer. per-step discr.	Sep. dec.
Embedding-space Diffusion LMs					
Diffusion-LM [Li et al., 2022]	DDPM	learn emb	Yes	Yes	
SED [Strudel et al., 2022]	DDPM	fix emb	Yes		
CDCD [Dieleman et al., 2022]	Score-ODE	learn emb	Yes		
DiffuSeq [Gong et al., 2023]	DDPM	learn emb	Yes	Yes	
GENIE [Lin et al., 2023]	DDPM	learn emb	Yes		
AR-Diffusion [Wu et al., 2023]*	DDPM	learn emb	Yes		
Plaid [Gulrajani and Hashimoto, 2023]	VLB	learn emb	Yes		
InfoDiffusion [Wang et al., 2023]	DDPM	learn emb	Yes		
Difformer [Gao et al., 2024]	DDPM	learn emb	Yes		
SeqDiffuSeq [Yuan et al., 2024]	DDPM	learn emb	Yes		
DINOISER [Ye et al., 2024]	SDE/DDIM	learn emb	Yes		
Simplex Diffusion LMs					
SSD-LM [Han et al., 2023]*	DDPM	simplex	Yes	Yes	
TESS [Mahabadi et al., 2024]	DDPM	simplex	Yes	Yes	
RDLM [Jo and Hwang, 2025]	RDM	simplex	Yes		
TESS 2 [Tae et al., 2025]	DDPM	simplex	Yes	Yes	
Latent Diffusion LMs					
LD4LG [Lovelace et al., 2023]*	DDPM	fix enc			Yes
PLANNER [Zhang et al., 2023]*	DDPM	fix enc			Yes
DGLM [Lovelace et al., 2024]*	VP-DDPM	fix enc			Yes
TEncDM [Shabalin et al., 2025]	VP-DDPM	fix enc			Yes
Cosmos [Meshchaninov et al., 2025]	VP-DDPM	fix enc			Yes
CoDAR [Shen et al., 2026]*	VP-SDE	fix enc			Yes
Flow-based LMs					
CFM [Roos et al., 2026]	FM	simplex	Yes		
FLM [Lee et al., 2026]	FM	one-hot	Yes		
DFM [Potaptchik et al., 2026]	FM	simplex	Yes		
LangFlow [Chen et al., 2026]	Bregman FM	learn emb	Yes		
ELF (ours)	**FM**	**fix enc**			

[†]*Process:* FM = Flow Matching [37, 38, 2]; DDPM = Denoising Diffusion Probabilistic Model [26]; VP-DDPM/-SDE = variance-preserving DDPM / stochastic differential equation [64]; Score-ODE = probability-flow ODE [64]; SDE/DDIM = continuous-time SDE [64] integrated with the deterministic DDIM solver; VLB = variational lower bound, specifically Plaid's $T \to \infty$ continuous-time limit [21]; RDM = Riemannian Diffusion Mixture, applied to the categorical sphere by RDLM [27]; Bregman FM = Flow Matching with a Bregman-divergence regression objective, used by LangFlow [10].

[‡]*State:* learn emb = jointly trained token embedding matrix; fix emb = frozen pretrained embedding lookup; fix enc = frozen pretrained encoder, optionally with a compressed autoencoder bottleneck on top; simplex = vocabulary-shaped logit simplex or square-root simplex on the sphere; one-hot = per-token one-hot stack over the vocabulary.

Table 2: **Survey of continuous diffusion and flow-based language models.** We summarize representative continuous diffusion and flow-based language models along several design axes. *Process* denotes the diffusion or flow process, with green indicating continuous-time formulations and red indicating discrete-time formulations. *State* denotes the continuous state in which denoising is performed. *Train per-step discr.* marks methods that convert intermediate denoising states to token predictions during training and apply token-level supervision such as cross-entropy loss at intermediate steps. *Infer. per-step discr.* marks methods that project intermediate sampling states back to token-aligned states during generation. *Sep. dec.* marks methods that require a separately trained decoder to map latent representations back to text. Blank entries indicate absence. * denotes autoregressive or block-autoregressive generation.

A Continuous Diffusion Language Model Survey

Survey details. We provide a detailed survey in Tab. 2. The survey summarizes representative continuous diffusion and flow-based language models along several design axes, including the underlying diffusion or flow process, the continuous state in which denoising is performed, whether intermediate denoising states are discretized during training or inference, and whether a separately trained decoder is required to map latent states back to text.

In particular, the *Train per-step discr.* and *Infer. per-step discr.* columns distinguish two different uses of intermediate discretization. *Train per-step discr.* indicates that intermediate denoising states are

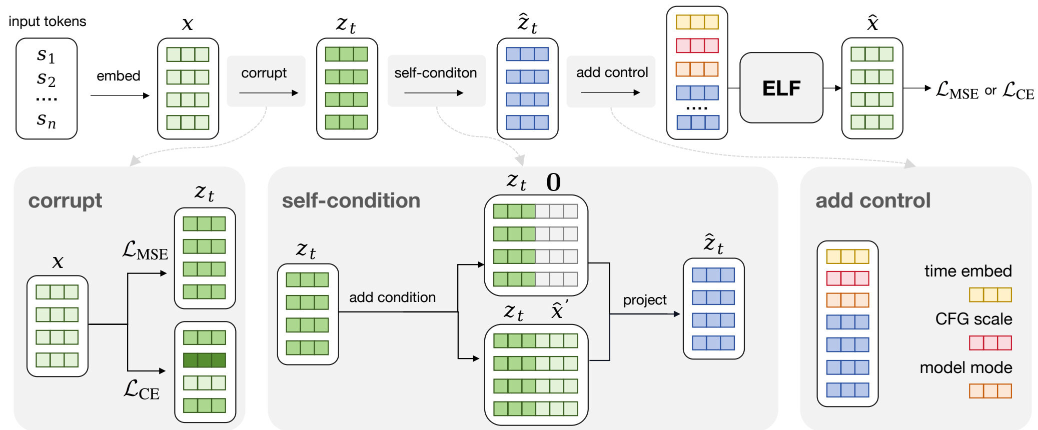

Figure 9: **Illustration of our training pipeline.** Starting from the clean embeddings $\boldsymbol{x}$, we apply different noise schedules in the two modes to obtain corrupted embeddings $\boldsymbol{z}_t$. We then apply self-conditioning by concatenating either $\mathbf{0}$ or the previous prediction $\hat{\boldsymbol{x}}'$ along the channel dimension, and project the concatenated embeddings back to the original dimension to form $\hat{\boldsymbol{z}}_t$. Next, we prepend control tokens to the embedding sequence, including time tokens in $[0, 1]$, CFG scale tokens in $[0.5, 5]$, and model-mode tokens indicating either denoising or decoding. The resulting sequence is fed into ELF to produce the final prediction $\hat{\boldsymbol{x}}$, which is supervised using either a denoising loss $\mathcal{L}_{\text{MSE}}$ or a token-wise cross-entropy loss $\mathcal{L}_{\text{CE}}$.

mapped to token predictions during training and supervised with token-level objectives such as cross-entropy loss. This provides direct vocabulary-level guidance, but also couples intermediate denoising states to categorical predictions. *Infer. per-step discr.* indicates that intermediate sampling states are explicitly projected back to token-aligned representations during generation, such as nearest-neighbor rounding in embedding space or argmax projection on a simplex. Methods without inference-time per-step discretization keep the sampling trajectory continuous and discretize only at the final step. The *Sep. dec.* column indicates whether a method requires a separately trained decoder to map continuous latent representations back to discrete text.

Positioning of ELF. Tab. 2 shows that existing continuous DLMs differ substantially in where the denoising process is defined and how continuous states are mapped back to text. Many embedding-space and simplex-based methods use training-time per-step discretization through token-level objectives, commonly cross-entropy, at intermediate denoising steps. These objectives provide direct token-level guidance, while making the denoising trajectory more tightly coupled to vocabulary-level prediction. Latent Diffusion LMs often avoid such per-step vocabulary supervision, but typically rely on DDPM-style or score-based formulations with DDPM noise schedules [26, 47] and require a separately trained latent-to-text decoder, such as an autoregressive decoder, non-autoregressive decoder, or latent decompressor, to recover discrete tokens.

ELF occupies a different design point. It formulates language generation as continuous-time Flow Matching in a frozen contextual embedding space and keeps the sampling trajectory continuous, applying discretization only at the final decoding step. Unlike prior latent Diffusion LMs, ELF does not require a separately trained decoder: a single shared-weight network performs intermediate denoising and recovers tokens at the final step through the unembedding layer.

B Method Details

B.1 Training

We show the full training pipeline in Fig. 9. The input tokens are first encoded into clean embeddings $\boldsymbol{x}$, which then go through three key steps before being fed into the ELF model: corruption, self-conditioning, and adding control tokens for conditioning and guidance. In the denoising branch, the model predicts clean embeddings $\hat{\boldsymbol{x}}$ and is supervised with $\mathcal{L}_{\text{MSE}}$. In the decoding branch, the same

Algorithm 3 ELF denoiser training with conditioning and guidance.

```
# net(z, t, c, w, mode): ELF network with in-context conditioning
# self_cond_proj(z): Self-conditioning projection layer that converts concatenated
    embeddings back to the original embedding dimension
# self_cond_prob: Self-conditioning probability
# s: a sequence of discrete tokens
# c: condition (only for conditional generation)

x = encode(s)
t = sample_t()
w = sample_sc_cfg_scale()
e = randn_like(x)
z = t * x + (1 - t) * e
v = x - e

# z w/o self-conditioning
z_no_sc = self_cond_proj(concat([z, zeros_like(z)], dim=-1))
x_no_sc = net(z_no_sc, t, c, w, mode="denoise")
v_no_sc = (x_no_sc - z) / (1 - t)

# z w/ self-conditioning
z_sc = self_cond_proj(concat([z, stopgrad(x_no_sc)], dim=-1))
x_sc = net(z_sc, t, c, w, mode="denoise")
v_sc = (x_sc - z) / (1 - t)

# Compute CFG target
v_target = v + (1 - 1 / w) * (v_sc - v_no_sc)

# Apply per-example self-conditioning mask
self_cond_mask = uniform(x.shape[0]) < self_cond_prob
v_pred = where(self_cond_mask, v_sc, v_no_sc)
v_target = where(self_cond_mask, v_target, v)
v_target = stopgrad(v_target)

# Compute v-loss
loss = mse_loss(v_pred, v_target)
```

shared-weight network predicts embeddings that are then passed through an unembedding layer and supervised with $\mathcal{L}_{\text{CE}}$. The full training algorithm is shown in Alg. 3 and Alg. 4.

Embedding corruption. First, we corrupt the clean embeddings $\boldsymbol{x}$ by adding noise. Specifically, we use $\boldsymbol{z}_t = t\boldsymbol{x} + (1-t)\boldsymbol{\epsilon}$ to obtain noisy embeddings $\boldsymbol{z}_t$, where $\boldsymbol{\epsilon}$ is Gaussian noise and t is the time step. Before corruption, we first normalize the clean embeddings using the estimated mean and standard deviation from the OWT dataset. We use different noise schedules for different modes.

For the denoising branch, we sample the time step t from a logit-normal distribution for each *sequence*. Specifically, we draw $t' \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ and map it to the unit interval via $t = \sigma(t')$, where $\sigma(\cdot)$ denotes the sigmoid function. In all experiments, we use $P_{\text{mean}} = -1.5$ and $P_{\text{std}} = 0.8$. We rescale the Gaussian noise by a factor of 2.

For the decoding branch, we train final-step discretization by conditioning the model on the decoder mode, *i.e.*, $t = 1$. At this time step, $\boldsymbol{z}_t$ corresponds to clean embeddings. Therefore, to make the final-step input nontrivial, we corrupt the clean embeddings with a per-token corruption level p sampled from a different noise schedule. Specifically, we draw p from a logit-normal distribution with $P_{\text{mean}} = 0.8$ and $P_{\text{std}} = 0.8$, and form $\tilde{\boldsymbol{z}} = p\boldsymbol{x} + (1-p)\boldsymbol{\epsilon}$, multiplying $\boldsymbol{\epsilon}$ by a noise scale. We use noise scales of 5 and 1 for OWT and conditional generation tasks, respectively. As a result, the corruption level varies across tokens within the same sequence. This design encourages the shared-weight decoder mode to recover corrupted embeddings from their surrounding context, making final-step discretization more robust to imperfect embeddings produced by the denoiser at inference time.

Algorithm 4 ELF decoder training with conditioning and guidance.

```
# net(z, t, c, w, mode): ELF network with in-context conditioning
# self_cond_proj(z): Self-conditioning projection layer that converts concatenated
    embeddings back to the original embedding dimension
# s: a sequence of discrete tokens
# c: condition (only for conditional generation)

x = encode(s)
p = sample_per_token_p()
w = sample_sc_cfg_scale()
e = randn_like(x)
z = p * x + (1 - p) * e

# use z w/o self-conditioning
z = self_cond_proj(concat([z, zeros_like(z)], dim=-1))
h = net(z, t=1, c, w, mode="decode")
s_pred = unembed(h)
loss = ce_loss(s_pred, s)
```

Self-conditioning. We apply self-conditioning following prior work [9]. During training, with a certain probability, we perform an additional forward pass to obtain the predicted embeddings $\hat{\boldsymbol{x}}'$, which are concatenated with the noisy embeddings $\boldsymbol{z}_t$ along the channel dimension. We stop the gradient through the predicted embeddings $\hat{\boldsymbol{x}}'$. For the remaining examples, we concatenate $\boldsymbol{z}_t$ with all-zero embeddings $\mathbf{0}$ instead. Since this concatenation doubles the channel dimension, we project it back to the original dimension using a linear layer. We apply self-conditioning with $\hat{\boldsymbol{x}}'$ in the denoising branch with 50% probability. For the decoding branch, we always use $\mathbf{0}$ as the self-conditioning input, as shown in Alg. 4.

Training-time CFG. As discussed in Sec. 3.3, our model performs training-time CFG [16, 17, 8, 69] with self-conditioning. In training-time CFG, the network is designed to model the post-combination quantity $\boldsymbol{v}_\theta^{\text{cfg}}$, rather than the pre-combination quantity $\boldsymbol{v}_\theta$. Following [16, 17], the regression target $\boldsymbol{v}_{\text{target}}$ is now:

$$\boldsymbol{v}_{\text{target}} = \boldsymbol{x} - \boldsymbol{\epsilon} + \left(1 - \frac{1}{\omega}\right)\big(\boldsymbol{v}_\theta^{\text{cfg}}(\boldsymbol{z}_t \mid t, \boldsymbol{c}, \omega) - \boldsymbol{v}_\theta^{\text{cfg}}(\boldsymbol{z}_t \mid t, \varnothing, \omega)\big), \tag{3}$$

where ω is the guidance scale. When $\omega = 1$, this reduces to the case without training-time CFG. In this case, the loss becomes $\|\boldsymbol{v}_\theta^{\text{cfg}}(\cdot) - \boldsymbol{v}_{\text{target}}\|^2$ [16, 17]. See Alg. 3. For each training example, we randomly sample a self-conditioning CFG scale $w \in [0.5, 5.0]$ from a power distribution biased toward smaller values [16, 17]. Since ELF uses $\boldsymbol{x}$-prediction, the quantity $\boldsymbol{v}$ is always converted from its $\boldsymbol{x}$ prediction counterpart (conditional or unconditional).

Our model uses a diverse set of conditions. Standard diffusion models typically implement conditioning through adaLN-Zero [50], which combines all conditioning signals through summation. This design becomes less effective when many heterogeneous conditions are present. Therefore, we adopt in-context conditioning [17] by prepending a set of *control* tokens that encode the conditioning information. Each control-token embedding has the same dimensionality as a standard language-token embedding. We prepend three types of control tokens: 4 time tokens with values in $[0, 1]$, 4 CFG-scale tokens sampled from $[0.5, 5]$, and 4 model-mode tokens indicating either denoising or decoding. These tokens are jointly trained with the model. All continuous values, *i.e.*, time and CFG scale, are encoded with positional embeddings.

For conditional generation, we place the clean embeddings of the conditioning sequence immediately after the control tokens and before the target sequence to be generated. The model then performs bidirectional self-attention over the concatenated sequence of conditioning and target tokens. The conditioning embeddings are kept uncorrupted during training. To enable CFG for conditional generation, we randomly drop the condition with 10% probability by zeroing out the embeddings of the conditioning sequence. This allows the model to learn both conditional and unconditional generation under the same framework.

Algorithm 5 ELF inference with conditioning and guidance.

```
# net(z, t, c, w, mode): ELF network with in-context conditioning
# self_cond_proj(z): Self-conditioning projection layer that converts concatenated
    embeddings back to the original embedding dimension
# shape: embeddings shape
# ts: discretized time grid over [0, 1] with N intervals
# c: condition (only for conditional generation)
# w: self-conditioning CFG scale

z = randn(shape)
x_pred = zeros(shape)

for i in range(len(ts) - 1):
    t = ts[i]
    dt = ts[i + 1] - ts[i]
    # Self-condition on the previous prediction
    z_sc = self_cond_proj(concat([z, x_pred], dim=-1))
    x_pred = net(z_sc, t, c, w, mode="denoise")
    # convert x prediction to velocity
    v = (x_pred - z) / (1 - t)
    z = z + dt * v

# decoding
z = self_cond_proj(concat([z, zeros_like(z)], dim=-1))
h = net(z, t=1, c, w, mode="decode")
# unembedding
token_logits = unembed(h)
tokens = argmax(token_logits)
```

B.2 Inference

We show the full inference algorithm in Alg. 5. Since the self-conditioning CFG scale is provided through in-context conditioning, changing w does not require an additional inference pass. By modifying w as a model input, we can flexibly control the trade-off between generation quality and diversity.

Time schedule. We discretize the continuous time interval $t \in [0, 1]$ into T intervals using a logit-normal time schedule. Specifically, we sample $T - 1$ time steps from the same logit-normal distribution used for the denoising branch during training and sort them to form the intermediate points. We use $P_{\text{mean}} = -1.5$ and $P_{\text{std}} = 0.8$ to match the training-time logit-normal distribution. We ensure that the first interval starts at $t = 0$ and the last interval ends at $t = 1$. This schedule produces smaller intervals when t is close to 0 and larger intervals as t approaches 1. It shows strong empirical performance, likely because the noisier regime requires finer discretization and the schedule better matches the noise distribution used during training.

Samplers. Our method supports both deterministic ODE sampling and an SDE-inspired stochastic sampler. The main algorithm in Alg. 2 uses the ODE sampler for simplicity, while Alg. 6 summarizes one-step updates for both samplers.

The SDE variant is motivated by the SDE associated with Flow Matching [43], whose dynamics can be interpreted as injecting infinitesimal noise at each step. In practice, we adopt a simple approximation that re-injects Gaussian noise at each sampling step while shifting the time variable slightly toward the noise regime. We introduce a noise re-injection scale γ to control the amount of stochasticity added at each step. The denoiser is then evaluated on this perturbed state, and its clean-embedding prediction is used to update the original state. When $\gamma = 0$, no stochastic perturbation is applied, and the update reduces to deterministic ODE sampling.

Algorithm 6 ELF inference with different samplers.

```
# z: noisy embeddings of current time step
# t: current time step
# dt: time interval, t_next - t
# gamma: controls the amount of noise added back in the SDE sampler

def ode_step(z, t, dt):
    x_hat = net(z, t, mode="denoise")
    v = (x_hat - z) / (1 - t)
    z = z + dt * v
    return z

def sde_step(z, t, dt, gamma):
    # Re-inject noise and move back to the corresponding time step
    # The jump size is defined relative to the time-step interval
    e = randn_like(z)
    alpha = 1 - gamma * dt
    t_back = alpha * t
    z_back = alpha * z + (1 - alpha) * e

    x_hat = net(z_back, t_back, mode="denoise")
    v = (x_hat - z) / (1 - t)
    z = z + dt * v
    return z
```

CFG for conditional generation. We apply standard CFG by combining the conditional and unconditional predictions. Similarly, we use the CFG scale to control the guidance strength.

C Additional Ablations

In this section, we present additional ablations of our design choices. Unless otherwise specified, all experiments use time schedule with either a 64-step ODE sampler or a 64-step SDE sampler with $\gamma = 1$. As before, we evaluate the generative perplexity–entropy trade-off by varying the self-conditioning CFG scale. We use red to indicate regions with poor generation quality, *i.e.*, entropy below 5.0, which often corresponds to repetitive or degenerate sentences, or generative perplexity above 300, which often corresponds to semantically meaningless or ungrammatical sentences. All models are trained for the same number of steps, with all other configurations kept the same as the default setting.

C.1 Prediction Targets

Our model directly predicts the clean embeddings $\boldsymbol{x}$ ($\boldsymbol{x}$-prediction). This allows us to use a unified denoiser and decoder through weight sharing and jointly optimize the model with both the denoising objective $\mathcal{L}_{\text{MSE}}$ and the token-level objective $\mathcal{L}_{\text{CE}}$. Prior work has also suggested that $\boldsymbol{x}$-prediction is essential, as high-dimensional clean data tends to lie on a low-dimensional manifold [32].

Here, we further study the effect of prediction targets. Specifically, since there are three quantities and two constraints: linear interpolation $\boldsymbol{z}_t = t\,\boldsymbol{x} + (1-t)\,\boldsymbol{\epsilon}$ and flow velocity $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{\epsilon}$, the network can be trained to predict one of these quantities, *i.e.*, $\boldsymbol{x}$-, $\boldsymbol{v}$-, or $\boldsymbol{\epsilon}$-prediction. To study this in a controlled setting, we use a two-stage pretrained encoder-decoder setup: a pretrained T5 encoder maps tokens into continuous embeddings, and a decoder is trained to reconstruct masked and noisy embeddings (See Sec. D.3 for details). We train only the denoising model while keeping both the encoder and decoder fixed. We use adaLN-Zero conditioning and a 64-step ODE sampler to plot the generative perplexity–entropy trade-off curve.

To study how prediction targets behave as the embedding dimension increases, we consider T5-small, T5-base, and T5-large encoders, corresponding to embedding dimensions of 512, 768, and 1024, respectively. We set the bottleneck dimension equal to the corresponding input embedding dimension.

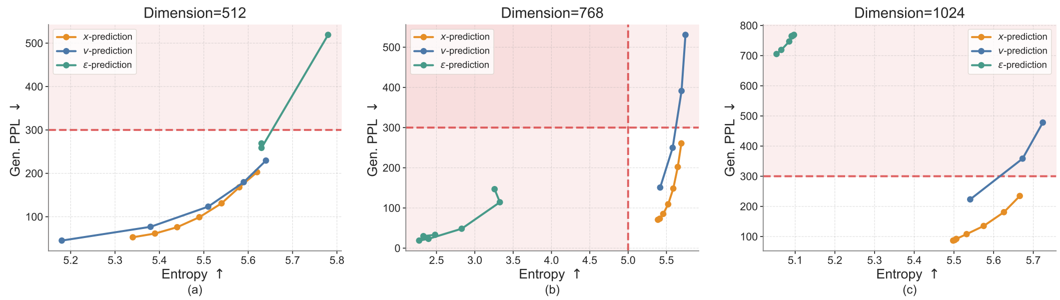

Figure 10: **Effects of prediction targets.** We vary the input dimension from 512 to 768 and 1024 by using T5-small, T5-base, and T5-large encoders, respectively. Across all input dimensions, $\boldsymbol{x}$-prediction remains stable and performs well. In contrast, $\boldsymbol{v}$-prediction performs well at 512 dimensions but degrades at higher dimensions, while $\boldsymbol{\epsilon}$-prediction collapses across all dimensions from 512 to 1024. The red region indicates poor-quality generations, where entropy falls below 5 (*e.g.*, repetitive sentences) or generative perplexity exceeds 300 (*e.g.*, meaningless or ungrammatical sentences). This aligns with the hypothesis from prior work that high-dimensional clean data often lies on a low-dimensional manifold [32].

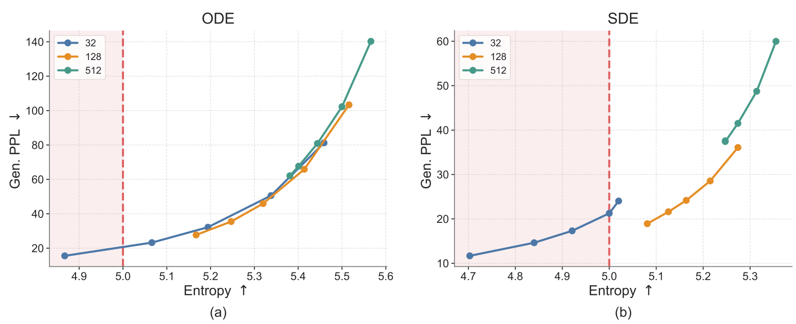

Figure 11: **Effect of bottleneck dimension.** We compare bottleneck dimensions of 32, 128, and 512 under ODE and SDE sampling. A moderate bottleneck dimension of 128 provides the best generative perplexity–entropy trade-off, while overly small or large bottlenecks either reduce diversity or hurt generative perplexity. Red indicates regions with poor generation quality, *i.e.*, entropy below 5.

As shown in Fig. 10, $\boldsymbol{x}$-prediction remains the most stable across all dimensions, maintaining a reasonable generative perplexity–entropy trade-off even at 1024 dimensions. In contrast, $\boldsymbol{v}$-prediction is competitive at 512 dimensions but degrades as the dimension increases, with substantially higher generative perplexity at 768 and 1024 dimensions. $\boldsymbol{\epsilon}$-prediction collapses across all dimensions, either achieving extremely low entropy or high generative perplexity, indicating repetitive, degenerate, or ungrammatical generations. These results support the hypothesis that clean-data prediction is better suited to high-dimensional language representations, consistent with findings from prior work [32].

C.2 Bottleneck

Our model uses a bottleneck design that projects encoder representations into a lower-dimensional space before mapping them back to the model hidden size. This design is motivated by the hypothesis that natural data may lie on a low-dimensional manifold within the high-dimensional embedding space. We compare bottleneck dimensions of 32, 128, and 512, and show the results in Fig. 11. The bottleneck dimension has a clear effect on the generative perplexity–entropy trade-off. Under ODE sampling, all three bottleneck sizes follow a similar frontier, but smaller bottlenecks tend to reach lower generative perplexity at the cost of lower entropy. Under SDE sampling, the differences become more significant: the 32-dimensional bottleneck achieves the lowest generative perplexity but often lies in the low-entropy region, indicating reduced diversity, whereas the 512-dimensional bottleneck maintains higher entropy but suffers from substantially worse generative perplexity. The 128-dimensional bottleneck provides the best overall balance, achieving strong generative perplexity while preserving reasonable entropy. We therefore use a bottleneck dimension of 128 as the default

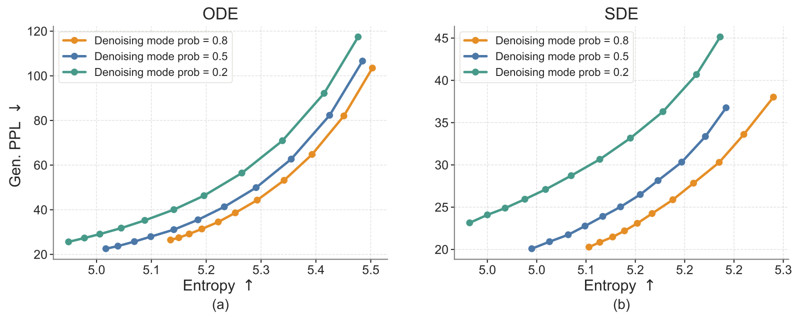

Figure 12: **Effect of the denoising mode probability during training.** This probability controls the allocation between denoising and decoding updates in the shared-weight denoiser-decoder model. A denoising mode probability of 0.8 provides the best generative perplexity–entropy trade-off across both ODE and SDE samplers.

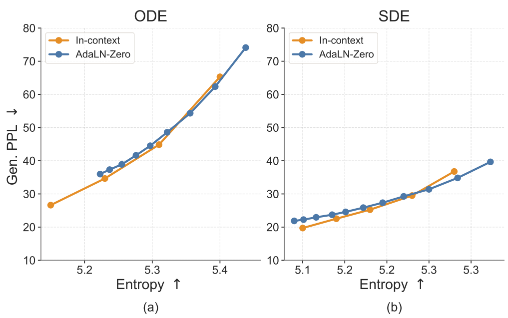

Figure 13: **Effect of conditioning strategies.** We compare in-context conditioning with adaLN-Zero conditioning. In-context conditioning slightly improves performance while substantially reducing the number of model parameters.

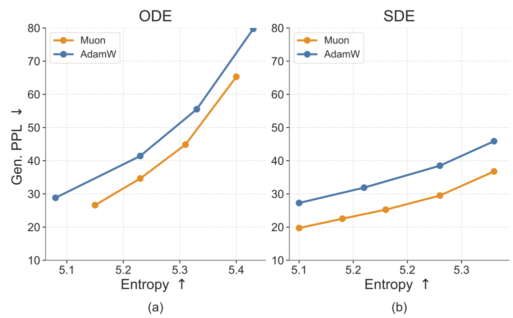

Figure 14: **Effect of optimizers.** We compare generation quality under different optimizers using Muon and AdamW. Muon achieves lower generative perplexity at comparable entropy under both ODE and SDE sampling methods.

setting. This finding is also consistent with prior work [32], which observes that an appropriate bottleneck can improve performance.

C.3 Denoising Mode Probability

Since ELF is trained with both MSE and CE losses through a shared-weight denoiser-decoder, each training step is assigned to either denoising mode or decoding mode. The denoising-mode probability controls this allocation: a higher probability emphasizes learning the continuous denoising dynamics, while a lower probability provides more supervision for mapping embeddings back to tokens. We study this trade-off by varying the denoising-mode probability during training.

As shown in Fig. 12, assigning a low probability to the denoising mode consistently degrades the generative perplexity–entropy trade-off, especially under SDE sampling. This suggests that the model requires sufficient training on the denoising process. Among the configurations tested, a denoising mode probability of 0.8 achieves the best overall trade-off across both ODE and SDE samplers. We therefore use 0.8 as the default denoising mode probability in our main experiments.

C.4 Conditioning Strategies

As discussed in Sec. 3.3, our model is conditioned on the time step, CFG scale, and model mode. We use in-context conditioning for these signals by prepending them as condition tokens to the input sequence, allowing the model to attend to them through full attention. This differs from the conventional adaLN-Zero conditioning design, which typically introduces additional model components to process the conditioning inputs. We compare these two designs in Fig. 13. In-context conditioning performs slightly better while avoiding the substantial parameter overhead introduced by

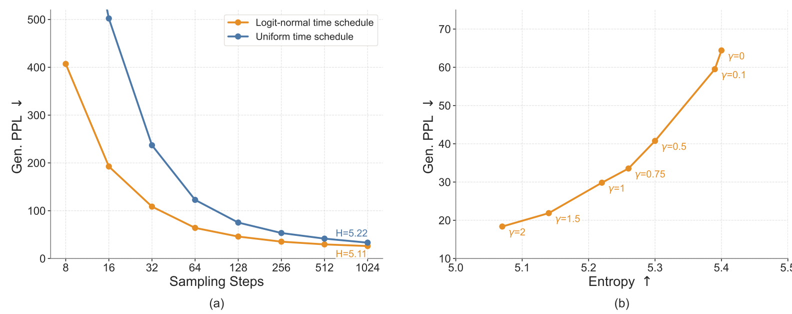

Figure 15: **Effect of time schedule and SDE noise re-injection scale.** (a) Logit-normal time schedule consistently improves generative perplexity across different sampling budgets, especially in the few-step regime. (b) The SDE noise re-injection scale γ controls the generative perplexity–entropy trade-off by adjusting the amount of stochastic noise injected during sampling.

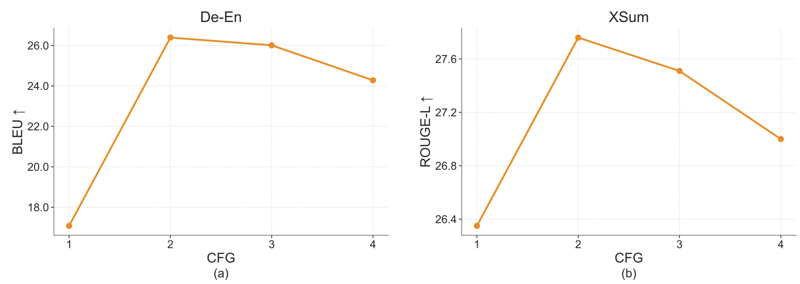

Figure 16: **Effect of CFG scale on conditional generation.** We sweep the CFG scale on WMT14 De-En translation and XSum summarization. Moderate guidance substantially improves task performance, with CFG scale 2 achieving the best result on both tasks, while overly strong guidance slightly degrades performance.

adaLN-Zero (ELF-B's parameter count is reduced from 148M to 105M). Therefore, we use in-context conditioning as our default setting.

C.5 Optimizers

We evaluate the impact of optimizer choice, comparing Muon [28] and AdamW [39], and show the results in Fig. 14. We tune the hyperparameters for both optimizers to obtain their best performance: for Muon, we use a learning rate of 2×10^{-3}; for AdamW, we use a learning rate of 1×10^{-4} with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. During training, Muon achieves lower loss within the same number of steps. During inference, models trained with Muon consistently achieve a better generative perplexity–entropy trade-off than those trained with AdamW under both samplers. The improvement is especially significant under SDE sampling, where Muon achieves lower generative perplexity at the same entropy level. These results highlight the importance of optimizer choice. Nevertheless, models trained with both optimizers still outperform other baselines, suggesting that the strong performance of ELF cannot be attributed to the optimizer alone.

C.6 Sampling Methods

We study two sampling design choices that improve inference efficiency and generation quality: sampling time schedule and stochastic SDE-inspired sampling. The logit-normal time schedule improves sampling efficiency by reducing the required number of denoising steps, while the SDE noise re-injection scale provides additional control over the generative perplexity–entropy trade-off.

Time schedules. By default, we use a logit-normal time schedule during inference [29]. We also evaluate an alternative uniform schedule. Fig. 15a shows the effect of the time schedule on ODE sampling across different numbers of sampling steps. Across all step counts, the logit-normal schedule consistently reduces generative perplexity compared with the uniform schedule. This improvement

Model	Depth	Hidden size	# Heads	Params	Training epochs
ELF-B	12	768	12	105M	5
ELF-M	24	1056	16	342M	4
ELF-L	32	1280	16	652M	3

Table 3: **ELF Model configurations** across different scales.

is especially significant in the few-step regime. These results suggest that the logit-normal time schedule improves sampling efficiency and final sample quality, likely because it better aligns the inference-time trajectory with the training-time schedule and allocates more sampling steps to noisier time steps.

SDE noise re-injection scale. For SDE sampling, we introduce a noise re-injection scale hyperparameter γ that controls the amount of stochasticity injected at each sampling step, as discussed in Sec. B.2. Intuitively, increasing γ introduces more stochasticity, while $\gamma = 0$ reduces to deterministic ODE sampling. As shown in Fig. 15b, γ controls the generative perplexity–entropy trade-off: within a moderate range, larger γ leads to lower generative perplexity while slightly reducing entropy. We hypothesize that the noise re-injection process helps correct early denoising errors, rather than deterministically amplifying imperfect trajectories as in ODE sampling. We therefore choose $\gamma = 1.0$ as our default setting, which provides a strong balance between generative perplexity and entropy.

C.7 CFG on Conditional Generation

We further study the effect of CFG scale on conditional generation tasks. As shown in Fig. 16, increasing the CFG scale from 1 to 2 substantially improves performance on both WMT14 De-En and XSum, suggesting that stronger conditioning helps the model better follow the source input. However, further increasing the scale leads to a gradual decline in performance, indicating that overly strong guidance can hurt generation quality. Based on this trend, we use CFG scale 2 as the default setting for conditional generation.

D Experimental Details

D.1 Model Architecture

Our model uses a standard Diffusion Transformer architecture [50]. We also incorporate popular general-purpose improvements, including SwiGLU [61], RMSNorm [80], RoPE [67], and qk-norm [24]. We use in-context conditioning instead of adaLN-Zero [50] conditioning, which allows us to significantly reduce the number of parameters; for example, the ELF-B model size is reduced from 148M to 105M parameters. Tab. 3 summarizes the configurations of ELF across different model sizes. We report the Transformer depth, hidden size, number of attention heads, and parameter count. We also report the number of training epochs used on the OWT dataset for each variant. Larger models tend to learn faster in our setup, and therefore require fewer training epochs.

D.2 Hyperparameters

ELF pipeline hyperparameters. Tab. 4 summarizes the main hyperparameters used in the ELF pipeline, covering model architecture, diffusion settings, conditioning and guidance, and optimization details. Unless noted otherwise, all experiments in the paper follow this default configuration. We include these settings for completeness and to facilitate reproducibility.

Inference-time settings for system-level comparison. For system-level comparison in Fig. 7, we use SDE sampling with time schedule enabled for all step budgets. We set the CFG scale to 3 for 8-, 16-, and 32-step generation. For SDE sampling, we use a stronger noise injection scale of $\gamma = 2$ in the very few-step regimes of 8 and 16 steps, and reduce it to $\gamma = 1.5$ for 32 steps, as longer denoising trajectories require less stochastic correction. For the system-level comparison in Tab. 1, we use 64-step ODE sampling with time schedule. We set the self-conditioning CFG scale to 1 and the input-condition CFG scale to 2.

Model Architecture		Denoising and Decoding Config	
Model	ELF-B	Time schedule	logit normal
Model size	105M	Denoiser $(P_{\text{mean}}, P_{\text{std}})$	$(-1.5,\ 0.8)$
Encoder backbone	T5-small	Denoiser noise scale	2.0
Embedding dimension	512	Decoder $(P_{\text{mean}}, P_{\text{std}})$	$(0.8,\ 0.8)$
Bottleneck dimension	128	Decoder noise scale	5.0
Model dimension	768	Denoiser *vs.* decoder prob.	0.8 *vs.* 0.2
Sequence length	1024		
Conditioning and Guidance		**Optimization and Training**	
Self-conditioning probability	0.5	Optimizer	Muon
Self-conditioning CFG range	$[0.5,\ 5]$	Learning rate	0.002
Num. of time tokens	4	Weight decay	0
Num. of model-mode tokens	4	Training epochs	5
Num. of CFG tokens	4	Global batch size	512
SDE γ	1.0	Learning rate schedule	constant
		Warmup epochs	0.5
		EMA decay	0.9999
		Training device	TPU v5p $\times$ 64
		Training time	1.5 h per epoch

Table 4: **Default training hyperparameters** and setup for ELF-B on the OpenWebText dataset. Unless noted otherwise, all experiments in the paper follow this default configuration.

Method	Base training	Distillation training	Effective tokens	Ratio
MDLM [56]	$512 \times 1\text{M} \times 1024$	-	524.3B	11.6×
Duo [57]	$512 \times 1\text{M} \times 1024$	-	524.3B	11.6×
MDLM + SDTT [56]	$512 \times 1\text{M} \times 1024$	$512 \times 10\text{K} \times 5 \times 1024$	550.5B	12.2×
Duo + DCD [57]	$512 \times 1\text{M} \times 1024$	$512 \times 10\text{K} \times 5 \times 1024$	550.5B	12.2×
FLM [30]	$512 \times 1\text{M} \times 1024$	-	524.3B	11.6×
FMLM [30]	$512 \times 1\text{M} \times 1024$	$512 \times 100\text{K} \times 1024$	576.7B	12.8×
LangFlow [10]	$512 \times 1\text{M} \times 1024$	-	524.3B	11.6×
ELF (ours)	$5 \times 9.04\text{B}$	-	**45.2B**	**1.0**×

Table 5: **Estimated effective training tokens** for ELF and the prior DLM baselines used in our system-level comparison (Fig. 7c). We estimate base-training tokens as batch size × steps × sequence length; distillation / flow-map stages are added on top where applicable.

Training-token budget for system-level comparison. Tab. 5 reports the estimated effective training tokens used by ELF and each baseline in Fig. 7c. We estimate base-training tokens as batch size × steps × sequence length and add distillation or flow-map stages on top where applicable. The OWT dataset contains roughly 9.04B tokens. With our default training schedule of 5 epochs, ELF therefore uses 45.2B effective training tokens. Thus, ELF requires roughly an order of magnitude fewer effective training tokens than the compared DLMs.

D.3 Ablation Studies Setting

We evaluate several choices of embedding representations for ELF, and report the implementation details as below. We also try two-stage training with a separate decoder. Unless specified, we keep other settings the same as the default ELF configuration.

Scratch encoder. We train an encoder from scratch on OpenWebText [18] by following the original T5-small training pipeline [53]. The encoder is trained for 5 epochs with a learning rate of 1×10^{-3}, cosine learning rate schedule, 0.4 epoch warmup, and a batch size of 512. During ELF training, we apply channel-wise normalization to the encoder outputs.

Steps	SC CFG	γ	Gen. PPL ↓	Entropy ↑
8	3	2.0	67.32 ± 2.25	5.14 ± 0.085
16	3	2.0	33.66 ± 1.09	5.16 ± 0.026
32	3	1.5	24.08 ± 0.16	5.15 ± 0.002

Table 6: **System-level ELF performance** reported as mean $\pm$ standard error (SE) over 6 independent evaluation runs (seeds 0–5; $n = 6$).

Sampler	SC CFG	ELF-B 105M		ELF-M 342M		ELF-L 652M	
		Gen. PPL	Entropy	Gen. PPL	Entropy	Gen. PPL	Entropy
SDE	0.5	36.77	5.28	39.21	5.35	37.50	5.41
	1.0	29.50	5.23	33.45	5.30	31.82	5.37
	1.5	25.25	5.18	28.42	5.26	28.72	5.35
	2.0	22.53	5.14	25.34	5.23	26.47	5.32
	3.0	19.72	5.10	21.69	5.18	23.31	5.28
	3.5	37.56	5.30	36.48	5.34	22.28	5.27
	4.0	36.50	5.29	34.93	5.33	21.37	5.26
ODE	0.5	104.29	5.51	88.51	5.51	68.27	5.52
	1.0	65.30	5.40	62.47	5.44	49.72	5.45
	1.5	44.85	5.31	46.71	5.37	39.97	5.40
	2.0	34.65	5.23	37.66	5.32	33.72	5.36
	3.0	26.62	5.15	28.80	5.24	26.57	5.29

Table 7: **Scaling performance** of generative perplexity (Gen. PPL) and unigram entropy for ELF models of different sizes under SDE and ODE samplers with 64 sampling steps. The effect of self-conditioning (SC) CFG scaling diminishes beyond 3.

Pretrained embedding layer. We use the frozen embedding table from the T5-small encoder as the token embedding layer. The embedding layer matrix is normalized, and the unembedding layer is trained separately.

Gaussian embedding layer. We randomly initialize and freeze an embedding layer from a Gaussian distribution, with token-wise embedding mean 0 and standard deviation 1. The unembedding layer is trained separately using the decoder mode.

Learnable embedding layer. We jointly train the embedding layer together with the denoiser and decoder modes. The unembedding layer is tied with the embedding layer: denoiser-mode updates affect the embedding layer, while decoder-mode updates affect the unembedding layer. To stabilize training, we apply normalization directly on the unembedding layer matrix at every step.

Separate decoder. For the separate-decoder setting, we use a randomly initialized decoder architecture obtained by mirroring the T5-small encoder. We keep the encoder fixed, mask 20% of the input tokens, and add logit-normal noise to the latent representations with $P_{\text{mean}} = 0.5$ and $P_{\text{std}} = 1.0$. The model is trained for 3 epochs with a learning rate of 3×10^{-4} and a cosine learning-rate schedule. The relative noise scale with respect to the normalized latent representations is set to 5.0.

D.4 Reported Numbers

System level comparison. Across 6 independent evaluation seeds, ELF shows highly consistent system-level behavior, as shown in Tab. 6. As the number of sampling steps increases from 8 to 32, the standard error (SE) decreases. The small standard errors—especially at 32 steps—suggest that these gains are robust to random seed variation and that the overall trend is reliable across runs. See Tab. 6 for detailed numbers.

Scaling behavior with CFG scales. The default setting for both sampling methods uses 64 sampling steps with time schedule. For the SDE sampler, we set $\gamma = 1.0$. The exact numbers are reported in

Config	AR	MDLM	E2D2	Duo	
Architecture					
Codebase	E2D2	E2D2	E2D2	Duo	Duo
Tokenizer	Qwen3-0.6B	Qwen3-0.6B	Qwen3-0.6B	T5-small	T5-small
Hidden Size	256	256	256	768	768
Intermediate Size	768	768	768	–	–
#Layers / Blocks	28	28	enc=20, dec=8	12	12
Sequence Length	64	64	64	64	64
Max Cond Length	1024	1024	1024	1024	64
Cond Embed	–	–	–	T5-small	T5-small
Training					
Dataset	XSum	XSum	XSum	XSum	De-En
Learning Rate	3e-4	3e-4	3e-4	3e-4	3e-4
LR Scheduler	const	const	const	const	const
Warmup Steps	1000	1000	1000	2500	2500
Global Batch Size	128	128	128	512	512
Optimizer	DecoupledAdamW	DecoupledAdamW	DecoupledAdamW	AdamW	AdamW
Loss Type	NLL	MDLM ELBO	E2D2 ELBO	Duo ELBO	Duo ELBO
Train Steps	500K	500K	500K	1M	1M
Evaluation					
Sampling Strategy	greedy	predict_and_noise	predict_and_noise	Duo sampler	Duo sampler
Sampling Steps	$L = 64$ (AR)	$\approx L$ (first-hit)	$\approx L$ (first-hit)	1000	1000
Block size	1	32	8	-	-
CFG Scale	–	–	–	1.0	1.5
Checkpoint	best	best	best	best	best
EMA	true	true	true	true	true

Table 8: **Detailed training and evaluation configurations for conditional generation tasks** of our reproduced AR, MDLM, E2D2, and Duo baselines. AR, MDLM, and E2D2 are reproduced on XSum using the E2D2 [4] codebase and follow the configurations reported in the E2D2 paper. For Duo, we build on the original Duo [57] repository, add cross-attention conditioning and CFG, adapt the T5-small encoder to match our setting, and tune the hyperparameters to obtain the strongest reproduced results.

Tab. 7. Larger CFG scales improve generation quality by reducing Gen. PPL within a certain range. The effect of CFG scaling reverses beyond 3. Only ELF-L benefits from increasing the CFG scale from 3 to 4. Thus, in most default ablation studies, we only consider CFG scales from 0.5 to 3.

D.5 Conditional Generation

Specifically, the WMT14 results for AR, MDLM, and E2D2 are taken from the E2D2 [4] paper, the SeqDiffuSeq result is taken from the LD4LG [41] paper, and the CDCD result is taken from the original CDCD [13] paper. For reproduced results, Duo [57] is implemented using the Duo codebase[4], while AR, MDLM, and E2D2 are reproduced using the E2D2 codebase[5].

For a fair comparison, we reproduce all baselines using settings that are as close as possible to their original implementations, as summarized in Tab. 8. For AR, MDLM, and E2D2, we use the E2D2 codebase and follow the training and evaluation configurations reported in the E2D2 paper on XSum. Note that although E2D2 is primarily designed for semi-autoregressive generation, we find that MDLM also achieves its best performance under a semi-autoregressive setting (*i.e.*, block size 32 with two-block generation); using single-block diffusion without semi-autoregressive generation degrades performance. For Duo, we start from the official Duo repository and adapt it to our conditional generation setting by adding cross-attention conditioning and classifier-free guidance, and by using a T5-small encoder for the conditioning input. During inference, we generate without

[4] https://github.com/s-sahoo/duo
[5] https://github.com/kuleshov-group/e2d2

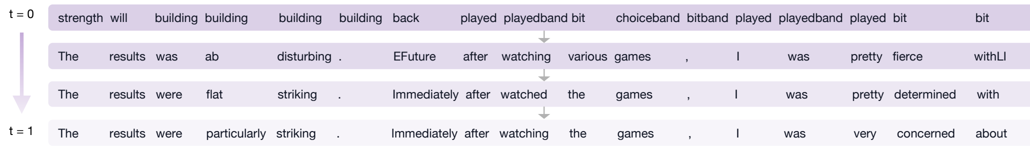

Figure 17: **Denoising trajectory** of ELF-B. As t increases from 0 to 1, ungrammatical sentences are progressively refined into fluent and grammatical text.

semi-autoregressive decoding. We tune the main sampling and guidance hyperparameters and report the best reproduced results we obtain.

E Qualitative Examples

E.1 Denoising Trajectory

Fig. 17 visualizes the intermediate predictions along ELF's denoising process. Starting from repetitive tokens at $t = 0$, the model gradually forms semantically meaningful phrases, improves grammar, and refines word choices as t approaches 1. This trajectory illustrates how continuous diffusion generation progressively transforms noisy embeddings that decode to gibberish text into clean embeddings that decode to grammatical sentences.

E.2 Unconditional Generation Examples on OpenWebText

We provide three unconditional samples generated by ELF-B on OpenWebText, reported with their entropy and generative perplexity (Gen. PPL). The examples illustrate that ELF produces fluent, syntactically coherent, and topically consistent long-form text across diverse domains.

ELF-B OWT

```
entropy: 5.36 Gen. PPL: 21.04

The company has been developing a virtual sleep mode for its iPhone and iPad
for years. This means that users can improve their quality of life without
turningping off their fingers thanks to Google's new virtual sleep technology.
To make the experience a reality, virtual sleep mode was developed for Google,
using a new built-in technology that includes real-time photography and shadow
monitoring. This technology enables users to have a safe, comfortable look at
where they sleep, even if you place the keyboard or a button under your
fingers. Some sources point to the iPhone 6 and iPhone 6 as yet another
example of the importance of virtual sleep mode in our everyday lives. This
technology has been shown to be useful when staying busy on tight days, during
difficult times or lying asleep on a hot night. This technology could also be
used to improve sleep quality and help users improve quality of life. Editor's
note: This post has been updated to answer to relevant questions. Google says
it will add virtual sleep mode to its iPhone and iPad in coming week. Google
announced some good news Thursday morning, including instructions on when to
eat, checking out, where to sleep and the rules surrounding what to eat. The
company reported a revenue of \$957 billion -- more than a third of the total
revenue during the same period. But the company doesn't seem to have a slew of
other good news yet, like the first one ...
```

ELF-B OWT

```
entropy: 5.27 Gen. PPL: 21.29

Balin said the potential cost of starting there is very low, and he told USA2
Network in an interview that he is not only interested in expanding the
capacity of the university, but is also interested in expanding other
```

```
services, including student assistance, community assistance, youth
assistance, youth assistance, and social justice assistance. Balin said, \"One
of the things about this is that it's difficult to start, because if you're
underfunded, you're going to need all the services that you need, and that's
what you have to pay for. And it's going to difficult, if nothing, for you to
get the funding you need to start right there.\" The UDU has not made such
promise. \"We have a lot of the guys in the department that are doing well, a
lot of the guys that aren't doing well at the university and they're currently
underfunded,\" Balin said. \"Most of the other LHS universities across the
country are currently underfunded. So, what do you want them to do? You know,
right now, the cost is very low and there are no great universities in the
rest of the country. It's not going to be easy.\" In the meantime, Balin said,
the UDU isn't looking to attract high-quality...
```

ELF-B OWT

```
entropy: 5.17 Gen. PPL: 21.80

Hey, I grew up in Lyndon in the early '90s and, after my father's death, began
writing a book about himself called The Life of Steve O'Malse. After his
second year at the University of Chicago O'Malse decided to write a biobio
about his father. He finished his study at the University of Chicago in the
fall of 1999. In 2009 he published a biobio called "My Dad And Daughter While
he Was Home," a successful biobio written by a former military officer, Lt.
Gen. David Wilde. Steve O'Malse has had great national security experience.
Throughout his career as a high-level national security adviser, he has served
as an adviser to George H.W. Bush and an adviser to two top FBI officials,
John J. Tillerson and Michael E. Comey, both of whom have been involved in the
investigations that led to the resignation of Attorney General Rex Tillerson.
He played a key role throughout the administration as a national security
adviser, then as a special adviser to former President George W. Bush, then as
secretary of Homeland Security under Bush and former presidential candidate
Ronald Clinton. In 2008, O'Malse was named by President Ronald Reagan as a new
national security adviser. In a speech last year, he detailed his experience
in the Reagan administration, as a new national security adviser. O'Malse said
he was surprised by his ability to express his concerns about national
security, but added that he would be speaking more for years to come...
```

E.3 Conditional Generation Examples

WMT14 De-En qualitative examples. We show qualitative examples on WMT14 De-En to complement the corpus-level BLEU results. ELF generally produces fluent and globally coherent translations.

ELF-B WMT-DE-EN

```
<Original text>

Dieses Phänomen hat nach den Wahlen vom November 2010 an Bedeutung gewonnen,
bei denen 675 neue republikanische Vertreter in 26 Staaten verzeichnet werden
konnten.

<Reference translation>

This phenomenon gained momentum following the November 2010 elections, which
saw 675 new Republican representatives added in 26 States.

<Our translation>

This phenomenon has increased in significance after the elections in November
2010, in which 675 new Republicanan representatives have been recorded in 26
countries.
```

ELF-B WMT-DE-EN

```
<Original text>

Im Gegensatz zu Kanada sind die US-Bundesstaaten für die Durchführung der
Wahlen in den einzelnen Staaten verantwortlich.

<Reference translation>

Unlike in Canada, the American States are responsible for the organisation of
federal elections in the United States.

<Our translation>

Unlike Canada, the United States are states responsible for holding elections
in each country.
```

XSum qualitative examples. We show qualitative examples on XSum to complement the ROUGE results. ELF generally produces fluent and concise summaries that capture the main content of the source document.

ELF-B XSum

```
<Original text>

Voges was forced to retire hurt on 86 after suffering the injury while batting
during the County Championship draw with Somerset on 4 June.
Middlesex hope to have the Australian back for their T20 Blast game against
Hampshire at Lord’s on 3 August.
The 37-year-old has scored 230 runs in four first-class games this season at
an average of 57.50.
"Losing Adam is naturally a blow as he contributes significantly to everything
we do," director of cricket Angus Fraser said.
"His absence, however, does give opportunities to other players who are
desperate to play in the first XI.
"In the past we have coped well without an overseas player and I expect us to
do so now."
Defending county champions Middlesex are sixth in the Division One table,
having drawn all four of their matches this season.
Voges retired from international cricket in February with a Test batting
average of 61.87 from 31 innings, second only to Australian great Sir Donald
Bradman’s career average of 99.94 from 52 Tests.

<Reference summarization>

Middlesex batsman Adam Voges will be out until August after suffering a torn
calf muscle in his right leg.

<Our summarization>

Middlesex captain Adam Voges will not miss the rest of the season as he
struggles with a bone injury.
```

ELF-B XSum

```
<Original text>

Ms Kendall told the BBC Labour risked sending a "resignation letter to the
British people as a serious party of government" by electing Mr Corbyn.
Separately, Ms Cooper warned there was a "serious risk the party will split"
if the left-winger becomes its leader.
It comes as Labour begins sending out the first ballot papers to voters.
```

The result of the contest will be announced at a special conference on 12 September.
More than 600,000 people have signed up to vote in the four-way contest but Labour has said applications are still being verified.
610,753
total electorate, though this may fall as party removes those not entitled to vote
Of which, full party members: 299,755
Affiliated to a trade union: 189,703
Registered to vote by paying GBP3: 121,295
Meanwhile voting in the election for the new Scottish Labour leader ended at midday.
Mr Corbyn is due to unveil a 10-point policy plan while in Glasgow later.
The popularity of the left-wing Islington North MP, who is promising "a new kind of politics", has sparked a row about the future direction of the Labour party.
Another leadership contender, Andy Burnham, told the BBC Mr Corbyn's policies "lack credibility".
"It's not possible to promise free university education, re-nationalising the utilities, without that coming at a great cost and if you can't explain how that is going to be paid for then I don't think we'll win back the trust of voters on the economy," he said.
BBC political correspondent Ross Hawkins said there had been "frustration" in rival camps who accused Mr Burnham of being reluctant to take on Mr Corbyn. This appeared to be his most direct attack yet, he added.
But in an interview with Jeremy Vine on BBC Radio 2, Mr Burnham declined to follow Ms Kendall and Ms Cooper and advise his supporters not to back Mr Corbyn with their second and third preferences.
He added: "People will say if they hear things like that, 'hang on, what do you believe?'"
In an interview with The Independent, Ms Kendall called for voters to mark Ms Cooper or Mr Burnham as second and third preferences, and avoid giving votes to Mr Corbyn.
"I have set out very clearly where I differ with all the candidates but our differences with Jeremy's kind of politics are far greater," said Ms Kendall.
Speaking on BBC Radio 4's Today programme she said she "can't pretend to be agnostic" about a victory for Mr Corbyn, saying of the voting process: "It is an alternative vote system and I want to urge party members to use all of their different preferences.
"I will be using my second and third preferences and I would urge others to do the same because I don't want to see our party go back to the politics of the '80s, just being a party of protest."
The Leicester West MP also said she did not see the party splitting, as it did in the 1980s when Labour members formed the Social Democratic Party.
However, Ms Cooper told BBC 2's Newsnight: "I think there is a serious risk that the party will split, will polarise and I cannot bear to see that happen because there is too much at stake."
Asked in an interview with grassroots Labour website Labourlist whether voters should use their votes to try to prevent Mr Corbyn winning, she said: "I think people should use all of their preferences.
"And I think the focus has to be how do we make sure we can win that election, and that's the most important thing - and I don't think Jeremy can do that."
Mr Corbyn has warned against "personal abuse" in the campaign, saying he wants to focus on policy.
His policy programme includes a commitment to "growth not austerity", nationalising the railways and energy sector, and a plan for nuclear disarmament.
In an essay for the Fabian Society he also suggested Labour's new increased following should be more involved in the party and proposed a review of membership fees to make the party more "inclusive".
Former Prime Minister Gordon Brown is expected to join the debate over the leadership contest with a speech on Sunday, called "power for a purpose - the future of the Labour Party".

Lance Price, former director of communications for Labour, told the BBC the contest had been an "unedifying mess" and had "done nothing to reengage the labour party with those millions of people who deserted it".
The Guardian newspaper has endorsed Ms Cooper for the leadership while the Daily Mirror has given its backing to Mr Burnham, although the paper urged him to "find a role" in his team for Mr Corbyn, who it says has "lit up the election campaign".
Labour leadership hopefuls Liz Kendall and Yvette Cooper have said their supporters should back anyone other than Jeremy Corbyn in the contest.

<Reference summarization>

Labour leadership hopefuls Liz Kendall and Yvette Cooper have said their supporters should back anyone other than Jeremy Corbyn in the contest.

<Our summarization>

Labour leadership contender Kendall Kendall has warned she does not avoid an threat of using Jeremy Corbyn with second and third preferences.