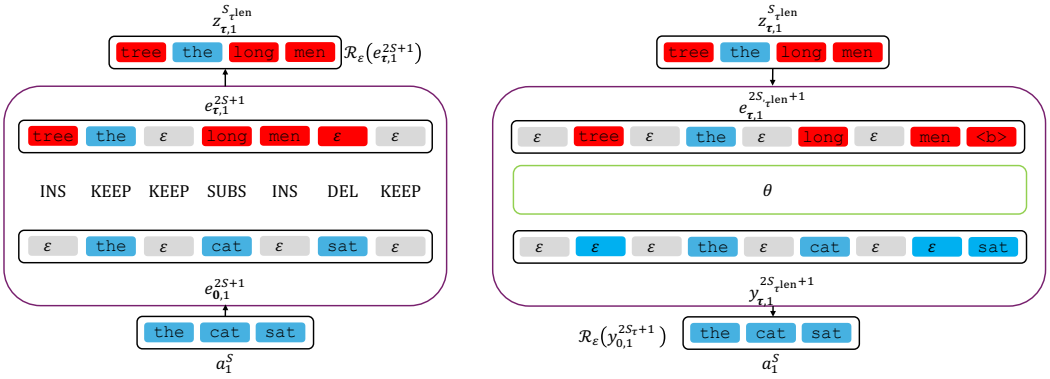
ERROR-AWARE FLEXIBLE-LENGTH DIFFUSION LANGUAGE MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models are trained to generate text by iteratively refining corrupted sequences in a non-autoregressive way. This makes them particularly attractive for error correction tasks. However, standard diffusion language models are usually trained on synthetically corrupted sequences and are limited to a fixed output length, which significantly constrains their performance. We propose a novel Flexible-Length Diffusion Language Model, which addresses the problem of fixed output length by allowing the model not only to substitute tokens during the reverse process but also to insert and delete them. Our model achieves a **21.25%** relative improvement over the best USDM configuration on HumanEval single-line code infilling, while maintaining comparable text generation performance. Moreover, we question whether random errors introduced by the standard forward process during training are sufficient for error correction. In particular, we study diffusion models as error correctors in automatic speech recognition (ASR) settings and introduce a framework for using TTS and CTC as a forward process to generate realistic ASR errors and train diffusion models with this task-specific corruption. Additionally, we extend existing methods for using diffusion models for ASR by proposing new decoding methods that not only significantly reduce the gap to autoregressive language models but also achieve a better WER-RTF trade-off. Our Flex ASR-DiffLM model achieves a WER of **3.41%** on LibriSpeech dev-other, surpassing the autoregressive language model on several evaluation sets while performing recognition **1.5-2x faster**. We publish all training and evaluation recipes.



(a) Forward process. (b) Reverse process.

Figure 1: Illustration of our Flexible-Length Diffusion Language Model.

1 INTRODUCTION

Discrete diffusion language models have emerged as promising alternatives to autoregressive models (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; 2025; Nie et al., 2025). They achieve similar generation performance while enabling parallel decoding. By learning to gradually denoise sequences, they are naturally suited for error correction. This has important applications in different

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

Table 1: Code-infilling performance of models. Oracle uses the ground-truth target length for initialization, except for variable-length Flex-DiffLM, where it provides the ground-truth number of deleted tokens in each gap.

Model	Initial Length					Avg.	Oracle
	4	8	16	32	64		
HumanEval Single-Line (Pass@1 [%]↑)							
MDLM	18.68	20.81	13.75	7.45	4.26	12.99	51.02
DreamOn	35.33	32.24	31.75	32.33	41.14	34.56	35.53
USDM	20.43	38.72	34.46	21.78	18.59	26.80	69.70
Flex-DiffLM (fixed length)	22.07	42.50	35.33	22.36	17.52	27.96	72.41
Flex-DiffLM	46.95					–	61.18
HumanEval Multi-Line (Pass@1 [%]↑)							
MDLM	4.28	4.92	4.11	2.05	1.03	3.28	13.84
DreamOn	8.00	7.96	7.46	7.74	9.89	8.21	8.51
USDM	4.56	9.03	11.90	10.21	6.96	8.53	28.08
Flex-DiffLM (fixed length)	4.99	9.85	11.83	10.51	7.19	8.87	29.05
Flex-DiffLM	12.88					–	21.60
SantaCoder-FIM (EM [%]↑)							
MDLM	15.34	9.97	12.75	12.94	14.77	13.15	37.87
DreamOn	32.50	29.53	30.58	30.49	41.04	32.83	31.26
USDM	17.35	16.49	24.07	27.13	27.90	22.59	51.77
Flex-DiffLM (fixed length)	17.45	15.72	22.91	24.54	27.13	21.55	50.72
Flex-DiffLM	45.45					–	45.45

fields such as coding (where code with syntactic or semantic errors should be corrected quickly) (Labs et al., 2025) and automatic speech recognition (ASR) (Leng et al., 2021; 2023), where acoustic models produce initial audio transcriptions with linguistic errors and a language model is used to refine them.

Limitations of Standard Diffusion LMs. However, standard diffusion LMs like USDM and MDLM face important limitations.

First, they cannot change sequence length during denoising, limiting their ability to correct insertions and deletions commonly found in ASR hypotheses, and to handle tasks where the target length is unknown, such as code infilling or text continuation. To address this problem, we propose a flexible-length diffusion model (Flex-DiffLM) that views sequence corruption in two dimensions: length noise τ^{len} and substitution noise τ^{sub}. At each reverse step, Flex-DiffLM places blank tokens between existing ones to allow new insertions, while existing tokens can be replaced or deleted. This enables correcting any edit error or generating text starting from an empty sequence without specifying the target length in advance, while allowing the denoising process to move along these two axes according to different schedules.

Second, there is a mismatch between synthetic training corruption and real-world hypotheses: during training, the USDM forward process randomly replaces tokens with vocabulary words, whereas ASR errors are typically acoustically similar to the correct ones and difficult for models trained only on synthetic noise to correct. We propose to replace standard synthetic corruption with an error-aware forward process: a text-to-speech (TTS) model generates audio from a clean sequence, and a CTC model decodes it into realistic ASR corruptions. Together, TTS and CTC form a task-specific forward process with which we train our fixed-length ASR-DiffLM and variable-length Flex ASR-DiffLM models.

We show that flexible-length denoising boosts code infilling, and layering realistic error training on top achieves a strong WER–RTF trade-off.

Contributions.

- We introduce a Flexible-Length Diffusion Language Model (Flex-DiffLM) that performs denoising in two dimensions: length and substitution noise. It achieves a 21.25% relative

improvement over the best USDM configuration on HumanEval single-line code infilling, while maintaining comparable text generation.

- We propose an error-aware forward process using TTS and CTC to generate realistic ASR errors. Trained on these errors, our Flex ASR-DiffLM achieves 3.41% WER on LibriSpeech dev-other, providing 1.5–2× faster decoding than both denoising and autoregressive LM baselines.
- We introduce three decoding methods for using diffusion LMs in speech recognition: DiffLM-Sum, Denoising with Rescoring, and Soft-Input Decoding. They effectively refine initial CTC hypotheses to achieve a better trade-off between recognition accuracy and decoding speed.
- We also propose a pseudo-log-likelihood sequence scoring method for diffusion LMs that is several times faster than existing approaches, matching standard LM sequence scoring in both speed and recognition accuracy.

2 PRELIMINARIES

Diffusion Language Models. Diffusion language models (DiffLMs) are generative models trained to denoise corrupted text. They are defined by two processes: forward and reverse (Austin et al., 2021). The forward process is a continuous-time Markov chain $\{z^S_{\tau,1}\}_{\tau\in[0,1]}$, where a^S_1 is a clean sequence of length S and $z^S_{\tau,1}$ is its corrupted version at noise level τ, with $z^S_{0,1} = a^S_1$. It is described by transition probabilities $p_{\text{fwd}}(z^S_{\tau',1} \mid z^S_{\tau,1})$ for $0 \le \tau < \tau' \le 1$, where τ denotes the noise level. This process gradually transforms the clean sequence $a^S_1 \sim p_{\text{data}}$ into a sequence from the noise distribution $z^S_{1,1} \sim p_{\text{noise}}$ by adding noise. The reverse process $p_{\text{bwd}}(z^S_{\tau,1} \mid z^S_{\tau',1}; \theta)$ is then learned to transform the corrupted sequence back to the clean sequence by iterative denoising. Usually, the neural network is trained to predict a^S_1 directly from $z^S_{\tau',1}$ at each denoising step by modeling $p_\theta(a^S_1 \mid z^S_{\tau',1}, \tau')$. This prediction is then plugged into the forward posterior to sample $z^S_{\tau,1}$ for the next denoising step. In this work, we consider two discrete diffusion language models: masked diffusion (MDLM) (Sahoo et al., 2024) and uniform-state diffusion (USDM) (Sahoo et al., 2025). MDLM corrupts sequences by replacing tokens with a special mask token and denoises them by predicting the masked tokens. USDM instead corrupts sequences by replacing tokens with uniformly sampled vocabulary tokens. During the reverse process, USDM re-evaluates every position, without an explicit indicator of which tokens are corrupted. This makes USDM naturally suited to error correction tasks, where corrupted positions are unknown during inference. Therefore, we focus primarily on this type of diffusion model and develop our methods based on its formulation.

3 FLEXIBLE-LENGTH DIFFUSION LANGUAGE MODEL

Different tasks require outputs with both the right content and the right length. We view this as denoising in two dimensions: one controls the tokens themselves through $\tau^{\text{sub}} \in [0, 1]$, while the other controls sequence length through $\tau^{\text{len}} \in [0, 1]$. We write $\boldsymbol{\tau} = (\tau^{\text{sub}}, \tau^{\text{len}})$ and $\boldsymbol{\tau}' = (\tau'^{\text{sub}}, \tau'^{\text{len}})$ for the current and next noise levels, with $\mathbf{0} = (0, 0)$. As discussed in Section 1, standard models such as USDM and MDLM move only along the first dimension, keeping the sequence length fixed. We propose a flexible-length diffusion language model (Flex-DiffLM), which can move along both dimensions.

3.1 FORWARD AND REVERSE PROCESSES

Forward process. We define a forward process that can dynamically change the length of the clean sequence a^S_1 through its direct corruption form $p_{\text{fwd}}(z^{S_{\tau^{\text{len}}}}_{\boldsymbol{\tau},1} \mid a^S_1)$. However, we do not apply the forward process directly to the observed sequence a^S_1. Instead, we extend our vocabulary with a special blank token ($V_{\text{blank}} = V \cup \{\varepsilon\}$) and construct an augmented sequence representation where these blank tokens are placed between consecutive original tokens. We refer to this interleaved structure as the *canvas*. We define the noise distribution p_{noise}, which consists only of the empty sequence. For a clean sequence a^S_1, the clean canvas is

$$e^{2S+1}_{\mathbf{0},1} = (\varepsilon, a_1, \varepsilon, a_2, \ldots, \varepsilon, a_S, \varepsilon)\,.$$

The forward process corrupts both types of positions: clean-token positions can be kept, substituted, or deleted, and gap positions can either stay empty or receive an inserted token, as illustrated in Figure 1a. This augmented representation makes the forward process similar to a mixture of absorbing-state and uniform-state diffusion models (von Rütte et al., 2025). Since all token and gap corruptions are independent given $\boldsymbol{\tau}$, the probability of the canvas factorizes as

$$p_{\text{fwd}}\left(e_{\boldsymbol{\tau},1}^{2S+1} \mid e_{\mathbf{0},1}^{2S+1}\right) = \prod_{s=1}^{2S+1} p_{\text{fwd},s}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right).$$

Each position uses a gap or token transition depending on its clean value:

$$p_{\text{fwd},s}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right) = \begin{cases} p_{\text{fwd}}^{\text{gap}}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right), & e_{\mathbf{0},s} = \varepsilon, \\ p_{\text{fwd}}^{\text{tok}}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right), & e_{\mathbf{0},s} \in V. \end{cases}$$

We use two noise levels: τ^{len} controls length-changing operations, while τ^{sub} controls substitutions. The probabilities used in our two-dimensional forward process are

$$p_{\text{del}} = \tau^{\text{len}}, \qquad p_{\text{sub}} = \tau^{\text{sub}}, \qquad p_{\text{ins}} = \tau^{\text{len}}\left(1 - \tau^{\text{len}}\right).$$

This choice ensures an empty sequence at $\boldsymbol{\tau} = (1, 1)$. The token and gap transitions are

$$p_{\text{fwd}}^{\text{tok}}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right) = \begin{cases} p_{\text{del}}, & e_{\boldsymbol{\tau},s} = \varepsilon, \\ \left(1 - p_{\text{del}}\right)\dfrac{p_{\text{sub}}}{|V|}, & e_{\boldsymbol{\tau},s} \in V \setminus \{e_{\mathbf{0},s}\}, \\ \left(1 - p_{\text{del}}\right)\left(1 - p_{\text{sub}} + \dfrac{p_{\text{sub}}}{|V|}\right), & e_{\boldsymbol{\tau},s} = e_{\mathbf{0},s}. \end{cases}$$

$$p_{\text{fwd}}^{\text{gap}}\left(e_{\boldsymbol{\tau},s} \mid e_{\mathbf{0},s}\right) = \begin{cases} 1 - p_{\text{ins}}, & e_{\boldsymbol{\tau},s} = \varepsilon, \\ \dfrac{p_{\text{ins}}}{|V|}, & e_{\boldsymbol{\tau},s} \in V. \end{cases}$$

The observed corrupted sequence, which we actually see during error correction and generation, is obtained by removing blank tokens from the canvas:

$$z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}} = \mathcal{R}_{\varepsilon}\left(e_{\boldsymbol{\tau},1}^{2S+1}\right),$$

where $\mathcal{R}_{\varepsilon}$ is a function that removes blank tokens. We can see that variable-length corruption is obtained through a fixed-length canvas followed by a collapse operation, as $S_{\tau^{\text{len}}}$ can be different from S. The probability of obtaining $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$ from the clean sequence a_1^S is obtained by summing over all canvases that collapse to $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$ after blank removal:

$$p_{\text{fwd}}\left(z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}} \,\middle|\, a_1^S\right) = \sum_{\substack{e_{\boldsymbol{\tau},1}^{2S+1}: \\ \mathcal{R}_{\varepsilon}\left(e_{\boldsymbol{\tau},1}^{2S+1}\right) = z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}}} p_{\text{fwd}}\left(e_{\boldsymbol{\tau},1}^{2S+1} \mid e_{\mathbf{0},1}^{2S+1}\right).$$

Reverse process. We parametrize the reverse process by training a Flex-DiffLM model p_θ to predict the clean sequence a_1^S from a corrupted sequence $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$. Similar to the forward process, our reverse process operates in the augmented representation of sequences, as shown in Figure 1b. However, during inference only $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$ is available, so we build the canvas from it and use it to predict the augmented representation of the clean sequence:

$$e_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1} = (\varepsilon, z_{\boldsymbol{\tau},1}, \varepsilon, z_{\boldsymbol{\tau},2}, \ldots, \varepsilon, z_{\boldsymbol{\tau},S_{\tau^{\text{len}}}}, \varepsilon).$$

The canvas $e_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$ built from $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$ and the canvas $e_{\mathbf{0},1}^{2S+1}$ built from the clean sequence a_1^S may have different lengths with high probability, so the clean canvas cannot serve as a position-wise target. Instead, we approximate the best possible correction of $e_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$ in one step: a target canvas $y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$ of the same length that brings the corrupted sequence as close as possible to the clean

sequence. This idea is similar to what was done by Dekel et al. (2026), but we use one sampled edit path as the training target, instead of marginalizing over all possible edit paths. Since the corrupted sequence is produced by the forward process during training, we know which operation produced each token $z_{\boldsymbol{\tau},i}$ and construct its target as follows, where i indexes the corrupted sequence and s indexes the clean sequence:

$$y_{\boldsymbol{\tau},2i} = \begin{cases} a_s, & z_{\boldsymbol{\tau},i} \text{ was produced by keeping or substituting } a_s, \\ \varepsilon, & z_{\boldsymbol{\tau},i} \text{ was produced by insertion.} \end{cases}$$

The interleaved blank positions $e_{\boldsymbol{\tau},2g+1}$, where $g \in \{0, \dots, S_{\tau^{\text{len}}}\}$ indexes the gaps in $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$, are used to predict clean tokens that were deleted by the forward process. Let $\mathcal{D}_{\tau^{\text{len}},g}$ denote the set of indices of clean tokens deleted at gap g in $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$. The target is

$$y_{\boldsymbol{\tau},2g+1} = \begin{cases} a_s, & s \sim \text{Uniform}(\mathcal{D}_{\tau^{\text{len}},g}), \quad \mathcal{D}_{\tau^{\text{len}},g} \neq \emptyset, \\ \varepsilon, & \mathcal{D}_{\tau^{\text{len}},g} = \emptyset. \end{cases}$$

At each reverse step, the model can recover only one deleted token per gap. A run of several deleted tokens can therefore be recovered over successive reverse steps, as every accepted insertion creates new gaps in the rebuilt canvas at the next step. By taking the argmax and removing blank tokens, we obtain our predicted clean sequence $\hat{a}_1^{\hat{S}}$:

$$\hat{a}_1^{\hat{S}} = \mathcal{R}_\varepsilon \left(\underset{y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}}{\arg\max}\, p_\theta \left(y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1} \;\middle|\; e_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}, \boldsymbol{\tau} \right) \right).$$

We parametrize the reverse step from $\boldsymbol{\tau}$ to $\boldsymbol{\tau}'$ as follows:

$$p_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta) = \begin{cases} p^{\text{gap}}_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta), & e_{\boldsymbol{\tau},i} = \varepsilon, \\ p^{\text{tok}}_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta), & e_{\boldsymbol{\tau},i} \in V. \end{cases}$$

For insertion slots, we draw a parallel with the MDLM reverse process: a deleted token remains absent during denoising, just as a masked token remains masked. Unlike a mask token, however, a blank can also be a valid clean target when no token needs to be inserted, or it can correspond to more than one token. Consider a gap containing $|\mathcal{D}_{\tau^{\text{len}},g}|$ clean tokens that were deleted at noise level τ^{len}. When moving to a lower noise level τ'^{len}, each of these tokens remains deleted with conditional probability $\tau'^{\text{len}}/\tau^{\text{len}}$. Since tokens are deleted independently, the probability that all $|\mathcal{D}_{\tau^{\text{len}},g}|$ tokens remain deleted is $(\tau'^{\text{len}}/\tau^{\text{len}})^{|\mathcal{D}_{\tau^{\text{len}},g}|}$. In this case, the insertion slot remains blank. Otherwise, at least one token should be restored, which occurs with probability $1 - (\tau'^{\text{len}}/\tau^{\text{len}})^{|\mathcal{D}_{\tau^{\text{len}},g}|}$.

$$p^{\text{gap}}_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta) = \left(\frac{\tau'^{\text{len}}}{\tau^{\text{len}}} \right)^{|\mathcal{D}_{\tau^{\text{len}},g}|} \mathbf{1}_{\{e_{\boldsymbol{\tau}',i} = \varepsilon\}} + \left(1 - \left(\frac{\tau'^{\text{len}}}{\tau^{\text{len}}} \right)^{|\mathcal{D}_{\tau^{\text{len}},g}|} \right) \bar{p}_{\theta,i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}).$$

Here, $\bar{p}_{\theta,i}$ is the probability distribution renormalized after excluding ε. For an existing token, we use the model's prediction to define a reverse transition that combines deletion and substitution:

$$p^{\text{tok}}_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta) = \begin{cases} r_{\text{del}}\, p_{\theta,i}(\varepsilon \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}), & e_{\boldsymbol{\tau}',i} = \varepsilon, \\ \big(1 - r_{\text{del}}\, p_{\theta,i}(\varepsilon \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau})\big) p^{\text{USDM}}_{\text{bwd},i}(e_{\boldsymbol{\tau}',i} \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau}; \theta), & e_{\boldsymbol{\tau}',i} \in V. \end{cases}$$

where

$$r_{\text{del}} = \begin{cases} \max\left(0,\, 1 - \dfrac{\tau'^{\text{len}}(1 - \tau'^{\text{len}})}{\tau^{\text{len}}(1 - \tau^{\text{len}})}\right), & 0 < \tau^{\text{len}} < 1, \\ 0, & \text{otherwise.} \end{cases}$$

At each reverse step, we decide whether a token inserted by the forward process should be deleted when moving from $\boldsymbol{\tau}$ to $\boldsymbol{\tau}'$. We delete it with probability $r_{\text{del}} p_{\theta,i}(\varepsilon \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau})$ and otherwise apply the USDM reverse transition for substitutions, since our forward substitution process is the same as in USDM. Similarly to the insertion transition, the factor r_{del} controls the probability of deletion as the length noise changes. The forward insertion probability $\tau^{\text{len}}(1 - \tau^{\text{len}})$ is non-monotonic: during the reverse process, it increases as τ^{len} moves from 1 to 0.5 and decreases from 0.5 to 0. We therefore clip negative values of the deletion factor to zero.

3.2 Training Objective

Our training objective combines the USDM ELBO for substitutions with cross-entropy losses for insertion, deletion and copy decisions. The copy term supervises positions where the model should keep an existing token slot or leave an empty gap unchanged. For $\boldsymbol{\tau} \sim \mathcal{U}([0,1]^2)$ and a corrupted sequence $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}} \sim p_{\text{fwd}}(\cdot \mid a_1^S, \boldsymbol{\tau})$, the objective is

$$\mathcal{L}_{\text{Flex-DiffLM}}(\theta) = \mathbb{E}_{\boldsymbol{\tau}, z_{\boldsymbol{\tau}}} \left[\frac{1}{S} \left(\mathcal{L}_{\text{sub}}^{\text{USDM}} + \mathcal{L}_{\text{ins}} + \mathcal{L}_{\text{del}} + \mathcal{L}_{\text{copy}} \right) \right].$$

Full definitions of all loss terms are provided in Appendix A.2.

3.3 Sampling

As in standard diffusion models, we can use ancestral sampling on a predefined grid of noise levels $\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_L = \mathbf{0}$, with $\boldsymbol{\tau}_l = (\tau_l^{\text{sub}}, \tau_l^{\text{len}})$ and non-increasing noise levels along each dimension. At each step, we sample the next state using p_{bwd} for the transition from $\boldsymbol{\tau}_l$ to $\boldsymbol{\tau}_{l+1}$. However, computing this transition requires $|\mathcal{D}_{\tau^{\text{len}},g}|$, which is unknown during inference. We therefore train a count head on top of our diffusion transformer to estimate the number of deleted tokens in each gap. Training and architecture details are in Appendix A.3. For substitution steps, we also adapt the ψ-sampler of Sahoo et al. (2025), which introduces additional stochasticity to allow the model to revise earlier token predictions.

Sampling flexibility: The two noise levels let us update length and substitutions jointly or separately. A transition from $(\tau^{\text{sub}}, \tau^{\text{len}})$ to $(\tau'^{\text{sub}}, \tau^{\text{len}})$ disables insertions and deletions and recovers the standard USDM sampling regime. A transition to $(\tau^{\text{sub}}, \tau'^{\text{len}})$ allows only insertions and deletions, leaving the remaining tokens unchanged.

4 Error-Aware Forward Process

Flex ASR-DiffLM. Given a clean sequence a_1^S, our goal is to produce a corrupted sequence $z_{\tau,1}^{S_\tau}$ that contains errors similar to those usually produced by ASR models. We use a text-to-speech (TTS) model to generate audio x_1^T from the sequence a_1^S and then an ASR model (in our case, CTC) to produce the noisy sequence $z_{\tau,1}^{S_\tau}$, as illustrated in Figure 4. As CTC can produce variable-length sequences, our main requirement for the forward process of our flexible diffusion framework is fulfilled. In contrast to synthetic corruption, however, we do not know which edit operations produced $z_{\tau,1}^{S_\tau}$ from a_1^S. We recover these operations using the Levenshtein edit distance algorithm (Algorithm 2), as illustrated in Figure 4b. The resulting edit path $\mathcal{P}_\tau$ transforms the clean sequence a_1^S into the CTC hypothesis $z_{\tau,1}^{S_\tau}$. We construct the interleaved input canvas as in Section 3.1 and use the recovered operations to construct the target canvas as in the reverse process in Section 3.1 (Algorithm 1). We train the model using the weighted cross-entropy objective defined in Appendix B.1 and call the resulting model Flex ASR-DiffLM.

Task-specific corruption: Using text-to-speech and speech-to-text models allows us to train on ASR-specific errors while preserving the flexible-length denoising capabilities of our model.

ASR-DiffLM. In addition, we train another model, ASR-DiffLM, with a similar TTS-CTC corruption process, but without allowing length changes. We align the CTC hypothesis with the clean sequence and reduce the alignment to its substitutions: clean tokens that would have to be inserted are dropped from the target, and hypothesis tokens that would have to be deleted are dropped from the input, leaving an equal-length pair in which every position is either a copy or a substitution. We do this to exclude the effect of flexible-length decoding and allow a fair comparison with USDM, so that we can evaluate whether an error-aware forward process improves recognition performance.

It is worth noting that we do not directly sample τ during training, and the corrupted sequence is therefore not conditioned on a sampled noise level. Instead, the noise level is implicitly determined

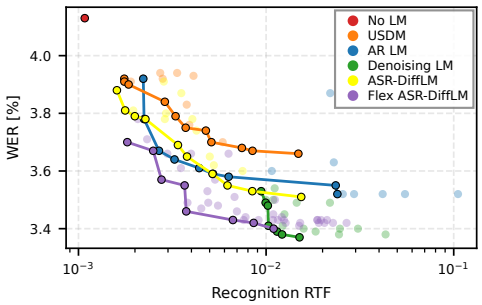


Figure 2: Speed-accuracy trade-off on dev-other with the TTS-trained CTC.

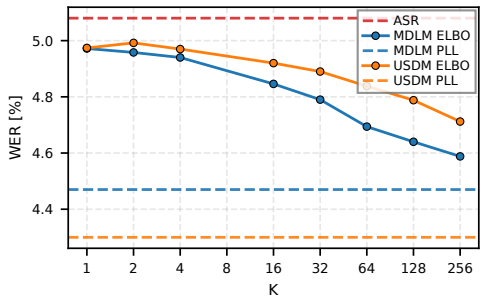


Figure 3: Rescoring on dev-other. K is the number of samples used in the MC estimate.

by the TTS-CTC corruption process, and ASR-DiffLM and Flex ASR-DiffLM are trained without time conditioning. In future work, we plan to make the TTS-CTC corruption explicitly depend on τ, for example by using τ to control dropout or other corruption parameters. We follow Koch et al. (2025); Gu et al. (2026) to train the TTS and CTC models for error generation.

5 Using Diffusion Language Models for ASR

Naveriani et al. (2026) introduced two ways to use diffusion language models for ASR: rescoring and joint decoding with CTC. In rescoring, the N most probable hypotheses from the ASR model are obtained and then rescored by the diffusion language model. Naveriani et al. (2026) estimate the score by Monte Carlo sampling, making rescoring much less efficient than with standard autoregressive language models. We address this gap with an efficient pseudo-log-likelihood score,

$$F_{\text{DiffLM-PLL}}(a_1^S) = \sum_{s=1}^{S} \log p_{\theta,s}(a_s \mid z_{0,1}^S = a_1^S, \tau = 0).$$

Using this efficient scoring function, we propose three correction-first decoding methods similar to those of Gu et al. (2026); Koch et al. (2025). DiffLM-Sum aggregates DiffLM predictions over several CTC hypotheses, which allows joint decoding to exploit more than one ASR hypothesis. Denoising with Rescoring applies denoising to CTC hypotheses and then rescores the union of the original and denoised candidates. Soft-Input Decoding replaces the discrete CTC hypotheses by a soft CTC posterior used as input to the model. Joint decoding and our methods can be viewed as error-correction methods: they use the diffusion LM to modify ASR hypotheses before selecting the final recognition output. The detailed definitions are provided in Appendix D.

6 Experimental Results

Experimental Setup. For speech recognition and text generation, we train on normalized LibriSpeech LM data and train-other transcriptions (∼900M tokens) (Panayotov et al., 2015). Diffusion models share a ∼350M-parameter DiT architecture (Peebles & Xie, 2023), trained for 10 epochs on one H100 GPU, except Flex-DiffLM, trained for 15 epochs on four GH200 GPUs. Our results show that this setup needs 1.5 times as many epochs for comparable performance. We mark Flex-DiffLM with * to indicate this different training setup. Single-GPU training is ongoing. Similarly sized AR and denoising LMs (Gu et al., 2026) are trained for 5 and 10 epochs, respectively, as longer training did not improve their performance. For code infilling, we follow Peng et al. (2025), training Qwen2.5-Coder-0.5B (Hui et al., 2024) with a diffusion objective for 5 epochs on a ∼1.1B-token Python subset, and use DreamOn's evaluation code (Wu et al., 2026). Training, TTS/CTC, and decoding details are in Appendices C and D.

Text generation. We evaluate Gen PPL, entropy, and MAUVE (Appendix E). As noted by Wang et al. (2025); Franca & Tong (2026), Gen PPL can favor simple, repetitive text, so MAUVE (Pillutla et al., 2021) is our primary quality metric. Flex-DiffLM performs comparably to MDLM and USDM (Table 2). Fixed-length Flex-DiffLM uses 96 substitution steps with $\tau^{\text{len}} = 0$. Variable-

Table 2: Unconditional generation and prefix continuation on LibriSpeech (1,500 samples per configuration). GPT-2 Large evaluates both perplexities. Mean length counts visible unconditional tokens. In the Length column, var. denotes variable-length generation. For Flex-DiffLM, +2 denotes initialization steps for variable-length generation only.

Model	Gen. steps	Length	Gen. PPL ↓	Mean length	Entropy ↑	MAUVE ↑	Cont. PPL ↓
Real text (dev-clean)	–	–	31.2	23.4	2.78	0.90	24.4
AR LM	–	16	40.1	16.0	2.65	0.15	13.6
		var.	36.2	15.3	2.40	0.86	13.1
MDLM	96	16	42.3	16.0	2.67	0.90	15.4
USDM			39.8	16.0	2.65	0.88	16.4
Flex-DiffLM	96 (+2)	16	40.5	16.0	2.66	0.91	16.4
		var.	40.8	14.7	2.49	0.84	16.0

length sampling alternates 32 length updates with two substitution steps each (hyperparameters in Appendix E.1). We also use two initial updates that insert a token into every gap to avoid short outputs averaging six tokens (Appendix G.1). Our continuation PPL scores generated sentence continuations (Appendix E.2.3). Diffusion models obtain similar PPL, slightly higher than that of AR. Fixed-length AR stops after exactly 16 tokens, often mid-sentence, helping explain its low MAUVE. Cropping reference and generated texts to 10 words raises MAUVE to 0.87.

Code infilling. Flex-DiffLM achieves the best non-oracle results among the evaluated diffusion models on all three benchmarks (Table 1) with one sampling configuration and no initial-length tuning. For MDLM and USDM, lengths that work well on one benchmark can hurt others. Like DreamOn (Wu et al., 2026) for masked diffusion, Flex-DiffLM enables variable-length infilling for USDM. On HumanEval single-line, it improves over USDM's average across lengths by 75.2%, while DreamOn improves over MDLM's average by 166.1%. Its 46.95% Pass@1 exceeds the best DreamOn result of 41.14% by 14.1% relatively. Qwen2.5-Coder initialization may benefit these objectives differently. Fixed-length Flex-DiffLM performs comparably to USDM. Variable-length Oracle replaces the count head with reference-derived insertion counts per gap.

Error-Correction Results. The joint decoding proposed by Naveriani et al. (2026) corrects a greedy CTC hypothesis with a diffusion LM, using acoustic evidence from CTC. In this setting, our error-aware models outperform the denoising LM (Table 5). Flex ASR-DiffLM achieves the lowest WER of 4.38%, compared with 4.56% for ASR-DiffLM and 4.69% for the denoising LM, while both diffusion models are much faster (RTF 0.0013 versus 0.0054). These results show that our models provide effective error correction using only a single greedy hypothesis. In Table 3, we evaluate error correction within an ASR system, where each language model is used together with a CTC model through the decoding methods described in Section 5. Our Flex ASR-DiffLM shows the best results among all diffusion models, achieving **3.41%** WER on dev-other and **3.77%** WER on test-other. Across decoding configurations (Figure 2), Flex ASR-DiffLM lies on the WER–RTF Pareto frontier, decoding **1.5–2 times faster** than AR and denoising LMs at comparable WER (baseline details in Appendix C). Error-aware training improves both fixed- and flexible-length models (Table 3), particularly with TTS-augmented CTC, while flexible-length modeling alone performs comparably to USDM. Longer training reaches **3.36%/3.69%** WER on dev-other/test-other after 25 epochs (Appendix G.3) (Prabhudesai et al., 2025; Ni et al., 2025). Without TTS, Flex ASR-DiffLM has no clear advantage over AR, which we leave for future analysis.

Table 5: Greedy correction combining CTC with a language model: WER and RTF on dev-other. The CTC acoustic model is trained without TTS data.

Language model	dev-other	
	WER [%] ↓	RTF ↓
None	5.08	0.0009
Denoising LM	4.69	0.0054
ASR-DiffLM	4.56	0.0013
Flex ASR-DiffLM	4.38	0.0013

Proposed Decoding Methods. Our methods improve over joint decoding for all models (Table 4). DiffLM-Sum works best for flexible-length models, while Denoising with Rescoring works best for fixed-length models using hypotheses of different lengths. PLL beats MC ELBO scoring for MDLM and USDM with one forward pass, while MC has not converged even at $K = 256$ (Figure 3). Its

Table 3: Best WER on the LibriSpeech evaluation sets. Each system is listed with its strongest decoding method. Scales are tuned on dev-other.

Model	Decoding method	WER [%] ↓			
		dev		test	
		clean	other	clean	other
CTC acoustic model without TTS data					
None	greedy	2.27	5.08	2.42	5.32
USDM	denoising + rescoring	1.99	4.22	2.13	4.60
Flex-DiffLM*	DiffLM-sum	2.08	4.32	2.21	4.79
AR LM	first-pass	1.85	3.85	2.01	4.20
Denoising LM	DLM-Sum	1.79	3.75	1.91	4.20
ASR-DiffLM	denoising + rescoring	1.85	4.07	2.06	4.56
Flex ASR-DiffLM	DiffLM-sum	1.80	3.88	1.95	4.36
CTC acoustic model with TTS data					
None	greedy	1.75	4.14	1.99	4.39
USDM	denoising + rescoring	1.63	3.66	1.82	3.96
Flex-DiffLM*	DiffLM-sum	1.64	3.75	1.88	4.02
AR LM	first-pass	1.58	3.52	1.77	3.72
Denoising LM	DLM-Sum	1.52	3.38	1.75	3.68
ASR-DiffLM	denoising + rescoring	1.59	3.53	1.82	3.89
Flex ASR-DiffLM	DiffLM-sum	1.50	3.41	1.77	3.77

Table 4: WER for different decoding methods with fixed- and flexible-length diffusion models, trained with synthetic corruption or ASR-induced errors. The CTC acoustic model is trained without TTS data.

Decoding method	WER [%] dev-other ↓			
	USDM	ASR-DiffLM	Flex-DiffLM*	Flex ASR-DiffLM
joint decoding (baseline)	4.65	4.56	4.68	4.38
soft-input greedy	4.63	4.50	4.64	4.35
DiffLM-sum	4.43	4.27	4.32	3.88
denoising + rescoring	4.22	4.07	4.37	3.96

success with MDLM suggests that it learns to evaluate unmasked tokens despite copying them during reverse sampling.

7 Related Work

Prior work such as DreamOn (Wu et al., 2026) and FlexMDM (Kim et al., 2026) enables variable-length generation with masked diffusion models. We choose DreamOn as our code-infilling baseline based on its authors' reported improvements over FlexMDM, and outperform it in our experimental setup (Table 1). DID (Ding et al., 2026) uses an insertion-only reverse process. Flex-DiffLM can also operate in this regime by keeping τ^{sub} fixed and disabling reverse deletions, while additionally supporting substitutions, which our findings show are crucial for ASR error correction. For error correction, we compare with the state-of-the-art denoising LM of Gu et al. (2026), achieving comparable recognition performance with faster decoding (Table 3 and Figure 2).

8 Conclusion and Limitations

In this work, we presented Flex-DiffLM, ASR-DiffLM, and Flex ASR-DiffLM, extending diffusion language modeling through flexible-length denoising and task-specific corruption. Flex-DiffLM improves HumanEval single-line performance by 21.25% over the best USDM configuration, while Flex ASR-DiffLM achieves 3.41% WER on LibriSpeech dev-other and 1.5–2× faster decoding than AR and denoising LMs at comparable WER. However, Flex-DiffLM permits only one insertion per gap per reverse step, and its canvas roughly doubles input length, increasing training cost.

AI USE STATEMENT

We used LLMs to help edit and rephrase text. For some parts of the paper, we also used them to draft initial text based on our own notes and outlines, ensuring that all main ideas and conclusions are our own. We additionally used LLMs to discuss the proposed formulation in Section 3 and help check our ideas, hypotheses, and mathematical reasoning. We used LLMs to find additional related work, but verified all suggested references ourselves. AI also assisted as a coding tool during experimental implementation. We take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the experimental details needed to reproduce our methods and evaluate them on speech recognition, text generation, and code infilling. Appendices A and B describe the training objectives, target construction, alignment procedure, and count head. Appendix C specifies the datasets, tokenizers, model architectures, and training hyperparameters. Appendix D details the ASR decoding methods, while Appendices E and F describe the sampling settings and evaluation procedures, including the metrics used to report our results. Additional experiments are presented in Appendix G.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

Avihu Dekel, Samuel Thomas, Takashi Fukada, and George Saon. Nle: Non-autoregressive llm-based asr by transcript editing. arXiv:2603.08397, 2026.

Fangyu Ding, Ding Ding, Sijin Chen, Kaibo Wang, Peng Xu, Zijin Feng, Haoli Bai, Kai Han, Youliang Yan, Binhang Yuan, and Jiacheng Sun. Beyond masks: Efficient, flexible diffusion language models via deletion-insertion processes. In *The Fourteenth International Conference on Learning Representations*, 2026.

Antonio Franca and Alexander Tong. Hacking generative perplexity: Why unconditional text evaluation needs distributional metrics. In *ICML 2026 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2026.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proc. ICML*, pp. 369–376, Pittsburgh, PA, USA, 2006.

Zijin Gu, Tatiana Likhomanenko, He Bai, Erik McDermott, Ronan Collobert, and Navdeep Jaitly. Revisiting asr error correction with specialized models. arXiv:2405.15216, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. arXiv:2409.12186, 2024.

Jaehyeon Kim, Sungwon Kim, Jungil Kong, and Sungroh Yoon. Glow-TTS: A generative flow for text-to-speech via monotonic alignment search. In *Advances in Neural Information Processing Systems*, 2020.

Jaeyeon Kim, Lee Cheuk Kit, Carles Domingo-Enrich, Yilun Du, Sham M. Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Samuel Albergo. Any-order flexible length masked diffusion. In *The Fourteenth International Conference on Learning Representations*, 2026.

Dorian Koch, Albert Zeyer, Nick Rossenbach, Ralf Schlüter, and Hermann Ney. Reproducing and dissecting denoising language models for speech recognition. arXiv:2512.13576, 2025.

Taku Kudo and John Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In *Proc. EMNLP: System Demonstrations*, 2018.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. arXiv:2506.17298, 2025.

Yichong Leng, Xu Tan, Linchen Zhu, Jin Xu, Renqian Luo, Linquan Liu, Tao Qin, Xiang-Yang Li, Edward Lin, and Tie-Yan Liu. Fastcorrect: fast error correction with edit alignment for automatic speech recognition. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 2021.

Yichong Leng, Xu Tan, Wenjie Liu, Kaitao Song, Rui Wang, Xiang-Yang Li, Tao Qin, Ed Lin, and Tie-Yan Liu. Softcorrect: error correction with soft detection for automatic speech recognition. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning*, 2024.

Vimal Manohar, Daniel Povey, and Sanjeev Khudanpur. Semi-supervised maximum mutual information training of deep neural network acoustic models. In *Interspeech 2015*, 2015.

Davyd Naveriani, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Diffusion language models for speech recognition. In *Interspeech 2026*, 2026.

Jinjie Ni, Qian Liu, Longxu Dou, Chao Du, Zili Wang, Hang Yan, Tianyu Pang, and Michael Qizhe Shieh. Diffusion language models are super data learners. arXiv:2511.03276, 2025.

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

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR Corpus Based on Public Domain Audio Books. In *Proc. IEEE ICASSP*, pp. 5206–5210, Brisbane, Australia, 2015.

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

Fred Zhangzhi Peng, Shuibai Zhang, Alex Tong, and contributors. Open-dllm: Open diffusion large language models, 2025. Blog: https://oval-shell-31c.notion.site/Open-Diffusion-Large-Language-Model-25e03bf6136480b7a4ebe3d53be9f68a?pvs=74, Model: https://huggingface.co/fredzzp/open-dcoder-0.5B.

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

Rohit Prabhavalkar, Takaaki Hori, Tara N Sainath, Ralf Schlüter, and Shinji Watanabe. End-to-end speech recognition: A survey. *IEEE/ACM Trans. Audio, Speech, and Language Processing*, 32: 325–351, 2023.

Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. In *Proc. NeurIPS*, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The Diffusion Duality. In *Proc. ICML*, 2025.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. arXiv:2104.09864, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Proc. NeurIPS*, 2017.

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

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas. In *The Fourteenth International Conference on Learning Representations*, 2026.

CONTENTS

1 Introduction . . . 1

2 Preliminaries . . . 3

3 Flexible-Length Diffusion Language Model . . . 3

3.1 Forward and Reverse Processes . . . 3

3.2 Training Objective . . . 6

3.3 Sampling . . . 6

4 Error-Aware Forward Process . . . 6

5 Using Diffusion Language Models for ASR . . . 7

6 Experimental Results . . . 7

7 Related Work . . . 9

8 Conclusion and Limitations . . . 9

A Flexible-Length Diffusion Language Model . . . 15

A.1 Target Canvas Construction . . . 15

A.2 Training Objective . . . 15

A.3 Predicting Deleted-Token Counts . . . 16

B Error-Aware Diffusion Language Model . . . 17

B.1 ASR-Aware Training Objective . . . 17

B.2 Alignment of ASR Errors . . . 17

C Experimental Details . . . 17

C.1 Data and Tokenization . . . 17

C.2 Language Models . . . 18

C.3 Code-Infilling Models . . . 19

C.4 TTS and ASR Models . . . 19

D Decoding Methods . . . 19

D.1 Rescoring . . . 19

D.2 Joint Decoding . . . 20

D.3 DiffLM-Sum . . . 21

D.4 Soft-Input Decoding . . . 21

D.5 Denoising with Rescoring . . . 21

D.6 Configuration . . . 22

E Generation and Evaluation 22
E.1 Sampling Settings 22
E.2 Evaluation Metrics 22

F Code-Infilling Evaluation 24

G Additional Experiments 24
G.1 Initialization 24
G.2 Fewer Sampling Steps 25
G.3 Longer ASR-Aware Training 25
G.4 Single- and Multi-GPU Training 25
G.5 Subword Segmentation Mismatch 26

H Generated Samples 26

A Flexible-Length Diffusion Language Model

We use the notation of Section 3: $\boldsymbol{\tau} = (\tau^{\text{sub}}, \tau^{\text{len}})$, $S_{\tau^{\text{len}}}$ is the corrupted sequence length, and i, s, and g index corrupted tokens, clean tokens, and gaps, respectively.

A.1 Target Canvas Construction

Algorithm 1 summarizes the target canvas construction procedure described in Section 3.1. When several clean tokens were deleted into one gap, we uniformly select one as the token target for that step. The total number of deleted tokens is retained separately for count-head supervision. Each edit-path entry records an operation o_k, a clean index s_k, and a corrupted index i_k; a missing index is written as $\varnothing$.

A.2 Training Objective

We name the position sets by the forward edit operations. Let $\mathcal{K}_{\boldsymbol{\tau}}$ contain token positions produced by keeping or substituting a clean token, $\mathcal{D}_{\boldsymbol{\tau}}$ contain gap positions at which a deleted clean token must be inserted, $\mathcal{I}_{\boldsymbol{\tau}}$ contain token positions produced by a forward insertion and therefore deleted during reconstruction, and $\mathcal{E}_{\boldsymbol{\tau}}$ contain empty gap positions at which no insertion is required. For $i \in \mathcal{K}_{\boldsymbol{\tau}} \cup \mathcal{D}_{\boldsymbol{\tau}}$, $\rho(i)$ denotes the index of the corresponding clean token, so $y_{\boldsymbol{\tau},i} = a_{\rho(i)}$. Here i indexes canvas positions. For compactness, we write $e_{\boldsymbol{\tau}} = e_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$ and $p_{\theta,i}(v) = p_{\theta,i}(v \mid e_{\boldsymbol{\tau}}, \boldsymbol{\tau})$.

Substitution loss. The model distribution over the original vocabulary is obtained by renormalizing after excluding the blank token:

$$\bar{p}_{\theta,i}(v) = \frac{p_{\theta,i}(v)}{1 - p_{\theta,i}(\varepsilon)}, \qquad v \in V.$$

We use this distribution in the continuous-time USDM substitution loss $\mathcal{L}_{\text{sub}}^{\text{USDM}}$ from Sahoo et al. (2025).

Input: Clean sequence a_1^S, corrupted sequence $z_{\boldsymbol{\tau},1}^{S_{\tau^{\text{len}}}}$, edit path $\mathcal{P}_{\boldsymbol{\tau}} = ((o_k, s_k, i_k))_{k=1}^{M_{\boldsymbol{\tau}}}$
Output: Target canvas $y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$

Initialize all positions of $y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$ with ε
$\mathcal{D}_{\tau^{\text{len}},g} \leftarrow \emptyset$ for each $g \in \{0, \ldots, S_{\tau^{\text{len}}}\}$
$g \leftarrow 0$
for $k = 1, \ldots, M_{\boldsymbol{\tau}}$ **do**
  if $o_k \in \{\text{KEEP}, \text{SUB}\}$ **then**
    $y_{\boldsymbol{\tau},2i_k} \leftarrow a_{s_k}$
    $g \leftarrow i_k$
  else if $o_k = \text{INS}$ **then**
    $y_{\boldsymbol{\tau},2i_k} \leftarrow \varepsilon$
    $g \leftarrow i_k$
  else if $o_k = \text{DEL}$ **then**
    $\mathcal{D}_{\tau^{\text{len}},g} \leftarrow \mathcal{D}_{\tau^{\text{len}},g} \cup \{s_k\}$
  end if
end for
for $g = 0, \ldots, S_{\tau^{\text{len}}}$ **do**
  if $\mathcal{D}_{\tau^{\text{len}},g} \neq \emptyset$ **then**
    Sample $s \sim \text{Uniform}(\mathcal{D}_{\tau^{\text{len}},g})$
    $y_{\boldsymbol{\tau},2g+1} \leftarrow a_s$
  end if
end for
return $y_{\boldsymbol{\tau},1}^{2S_{\tau^{\text{len}}}+1}$

Algorithm 1: Constructing the target canvas from an edit path

Insertion, deletion and copy losses. The remaining terms are standard cross-entropy losses. The insertion and deletion losses are

$$\mathcal{L}_{\text{ins}} = -\sum_{i \in \mathcal{D}_{\boldsymbol{\tau}}} \log p_{\theta,i}\left(a_{\rho(i)}\right), \tag{1}$$

$$\mathcal{L}_{\text{del}} = -\sum_{i \in \mathcal{I}_{\boldsymbol{\tau}}} \log p_{\theta,i}(\varepsilon). \tag{2}$$

At a surviving token position, any non-blank prediction preserves the token slot. At an empty gap position, predicting the blank token leaves the gap unchanged. Therefore, the copy loss is

$$\begin{aligned}\mathcal{L}_{\text{copy}} = &-\sum_{i \in \mathcal{K}_{\boldsymbol{\tau}}} \log\left(1 - p_{\theta,i}(\varepsilon)\right) \\ &-\sum_{i \in \mathcal{E}_{\boldsymbol{\tau}}} \log p_{\theta,i}(\varepsilon).\end{aligned} \tag{3}$$

The first term supervises whether an existing token slot should be preserved, while the particular token that should be at that position is supervised separately by $\mathcal{L}_{\text{sub}}^{\text{USDM}}$.

A.3 Predicting Deleted-Token Counts

The reverse insertion probability depends on the number of clean tokens deleted into each gap. This number is available from the forward edit path during training, but is unknown during generation. We predict it with a separate count head using the hidden states of the trained Flex-DiffLM.

Architecture. The head applies two one-dimensional convolutional layers with 384 channels and a kernel size of 3 to the interleaved hidden-state sequence. Each layer uses ReLU, layer normalization, and a dropout rate of 0.1, followed by a linear projection to a scalar. We read predictions at gap positions. During count-head training, the backbone is frozen with dropout disabled, while dropout remains active in the count head at a rate of 0.1.

Objective. We train the count head using mean squared error (MSE). The target count for gap g is $|\mathcal{D}_{\tau^{\text{len}},g}|$, as defined in Section 3.1. Let $h_{\boldsymbol{\tau}}$ be the backbone hidden-state sequence for $e_{\boldsymbol{\tau}}$ and $c_\phi(h_{\boldsymbol{\tau}})_g$ the head's prediction at canvas position $2g+1$. For editable gaps $\mathcal{G}_{\boldsymbol{\tau}}$, we minimize

$$\mathcal{L}_{\text{count}}(\phi) = \frac{1}{|\mathcal{G}_{\boldsymbol{\tau}}|} \sum_{g \in \mathcal{G}_{\boldsymbol{\tau}}} \left(c_\phi(h_{\boldsymbol{\tau}})_g - \min\left(|\mathcal{D}_{\tau^{\text{len}},g}|, 15\right)\right)^2.$$

We heuristically choose a count cap of 15 to reflect typical missing-span lengths in code infilling and text continuation, and clip training targets to this value. We did not tune this parameter, as the count head already performed well on the coding benchmarks. This per-gap cap does not impose a maximum output length, since each insertion creates new gaps. At inference, predictions are rounded and clipped to $\{0, \ldots, 15\}$. The predicted count controls the probability of inserting a token, while the backbone predicts its identity. Each gap still permits at most one insertion per reverse step. For text generation, however, we observe that the count head often underestimates the number of missing tokens, leading to short sequences. We leave improving the count head for text generation to future work.

We also observed that conditioning on τ^{len} can leak information about the number of deleted tokens to the count head, causing it to rely more on the noise level than on the sequence context. We therefore use the true τ^{len} for backbone conditioning with probability 0.5 and set the conditioning value to zero otherwise. In both cases, the sequence is corrupted using the true τ^{len}.

Training for text generation. The selected head is trained on the LibriSpeech text data with an equal mixture of whole-sequence corruption and corruption restricted to a random internal span. Outside the span, tokens remain unchanged. We use AdamW with weight decay 0.01, gradient clipping at norm 1, and the piecewise linear learning-rate schedule in Appendix C.2. Training uses bfloat16, packed tensors and CUDA graphs on four NVIDIA GH200 GPUs, with a batch budget of 10,000 text tokens per GPU and 20 partitioned training subepochs. The selected checkpoint is the final head checkpoint. Its frozen backbone is the same checkpoint used for text generation.

Inference. For the main variable-length generation configuration, the head reuses the hidden states of each length update, so it requires no additional backbone evaluation. It is evaluated again after the canvas changes. Oracle code-infilling runs replace predicted counts with reference-derived counts and are reported separately.

B Error-Aware Diffusion Language Model

B.1 ASR-Aware Training Objective

As discussed in Section 4, we do not sample τ and train our models without noise-level conditioning. For simplicity, we use τ in our notation to denote the noise level implicitly defined by the ASR model. We train both ASR-aware models using weighted cross-entropy. For Flex ASR-DiffLM, the objective is

$$\mathcal{L}_{\text{Flex ASR-DiffLM}}(\theta) = \frac{-\sum_{i=1}^{2S_\tau+1} w_{\tau,i} \log p_{\theta,i}\left(y_{\tau,i} \,\middle|\, e_{\tau,1}^{2S_\tau+1}\right)}{\sum_{i=1}^{2S_\tau+1} w_{\tau,i}},$$

where

$$w_{\tau,i} = \begin{cases} \lambda_{\text{copy}}, & e_{\tau,i} = y_{\tau,i}, \\ 1, & e_{\tau,i} \neq y_{\tau,i}. \end{cases}$$

For ASR-DiffLM, let $\tilde{z}_1^{\tilde{S}}$ and $\tilde{a}_1^{\tilde{S}}$ denote the reduced, equal-length input and target sequences constructed in Section 4. We apply the loss directly to these sequences without interleaving blank positions:

$$\mathcal{L}_{\text{ASR-DiffLM}}(\theta) = \frac{-\sum_{s=1}^{\tilde{S}} \tilde{w}_s \log p_{\theta,s}(\tilde{a}_s \mid \tilde{z}_1^{\tilde{S}})}{\sum_{s=1}^{\tilde{S}} \tilde{w}_s}, \qquad \tilde{w}_s = \begin{cases} \lambda_{\text{copy}}, & \tilde{z}_s = \tilde{a}_s, \\ 1, & \tilde{z}_s \neq \tilde{a}_s. \end{cases}$$

We use $\lambda_{\text{copy}} = 0.1$ for both models.

We have not evaluated iterative denoising with models trained using the ASR-aware forward process. Our decoding methods use only the model's clean-token predictions, which do not require a count head.

B.2 Alignment of ASR Errors

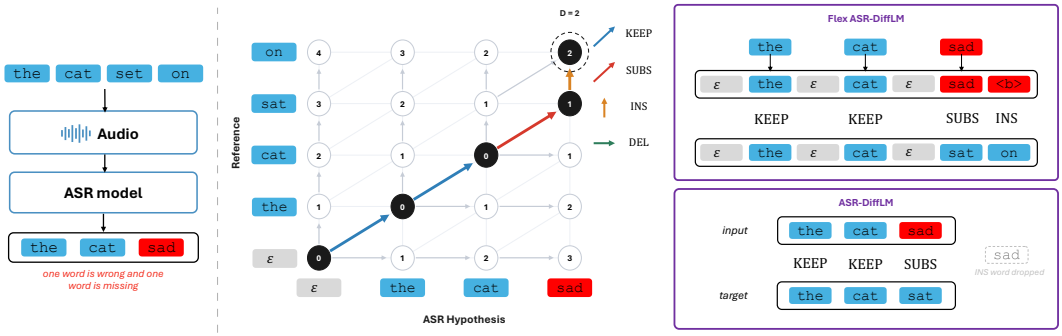
We align the corrupted CTC hypothesis with the clean reference using Levenshtein distance. Algorithm 2 returns the token-level edit path used by Algorithm 1 to construct the target canvas. For this application of Algorithm 1, we replace $\boldsymbol{\tau}$ by τ, $S_{\tau^{\text{len}}}$ by S_τ, and $\mathcal{D}_{\tau^{\text{len}},g}$ by $\mathcal{D}_{\tau,g}$. Before token-level alignment, we reconcile alternative subword segmentations of correctly recognized words as described in Appendix G.5.

C Experimental Details

C.1 Data and Tokenization

For speech recognition and text generation, all language models were trained on the normalized LibriSpeech LM corpus (Panayotov et al., 2015) together with train-other transcriptions (∼900M tokens and ∼800K distinct words). We used a SentencePiece tokenizer with 10,240 subwords (Kudo & Richardson, 2018). For ASR evaluation, we use dev-clean, dev-other, test-clean and test-other, with all scales tuned on dev-other and kept fixed for the other evaluation datasets. All models were trained with the AdamW optimizer with a weight decay of 0.01 (Loshchilov & Hutter, 2019). We used a piecewise linear learning-rate schedule, with learning rates of 10^{-5}, 10^{-3}, 10^{-5} and 10^{-6} at 0%, 45%, 90% and 100% of training, respectively. For the synthetic-corruption Flex-DiffLM and its count head, we restrict clean text to at most 75 tokens.

For code infilling, we use part of the Python subset of the training dataset of Peng et al. (2025), comprising approximately 1.1B tokens. We use the Qwen2.5-Coder tokenizer (Hui et al., 2024), with a vocabulary of 151,665 tokens. We concatenate documents with EOS separators and split the resulting token stream into blocks of 4,096 tokens, discarding incomplete trailing blocks.



(a) Error-aware forward process.

(b) Training ASR Error-Aware Diffusion LMs.

Figure 4: Overview of the forward and training processes for Flex ASR-DiffLM and ASR-DiffLM.

Input: Clean sequence a_1^S, corrupted sequence $z_{\tau,1}^{S_\tau}$
Output: Edit path $\mathcal{P}_\tau$ from a_1^S to $z_{\tau,1}^{S_\tau}$

$D_{s,0} \leftarrow s$ for $s = 0, \ldots, S$; $D_{0,i} \leftarrow i$ for $i = 0, \ldots, S_\tau$
for $s = 1, \ldots, S$ **do**
  for $i = 1, \ldots, S_\tau$ **do**
    $c_{s,i} \leftarrow \mathbf{1}_{\{a_s \neq z_{\tau,i}\}}$
    $D_{s,i} \leftarrow \min\{D_{s-1,i-1} + c_{s,i},\ D_{s-1,i} + 1,\ D_{s,i-1} + 1\}$
  end for
end for
$s \leftarrow S$; $i \leftarrow S_\tau$; $\mathcal{P}_\tau \leftarrow [\,]$
while $s > 0$ or $i > 0$ **do**
  if $s > 0$ and $i > 0$ and $D_{s,i} = D_{s-1,i-1} + c_{s,i}$ **then**
    if $a_s = z_{\tau,i}$ **then**
      Prepend (KEEP, s, i) to $\mathcal{P}_\tau$
    else
      Prepend (SUB, s, i) to $\mathcal{P}_\tau$
    end if
    $s \leftarrow s - 1$; $i \leftarrow i - 1$
  else if $s > 0$ and $D_{s,i} = D_{s-1,i} + 1$ **then**
    Prepend $(\text{DEL}, s, \varnothing)$ to $\mathcal{P}_\tau$
    $s \leftarrow s - 1$
  else
    Prepend $(\text{INS}, \varnothing, i)$ to $\mathcal{P}_\tau$
    $i \leftarrow i - 1$
  end if
end while
return $\mathcal{P}_\tau$

Algorithm 2: Levenshtein edit distance

C.2 Language Models

All language models except Flex-DiffLM were trained on a single NVIDIA H100 GPU, with 80 GB of memory for ASR-DiffLM and Flex ASR-DiffLM and 96 GB for the standard LM, denoising LM, MDLM and USDM. Flex-DiffLM was trained on four NVIDIA GH200 GPUs, each with 96 GB of memory.

For the main comparisons, diffusion models are trained for 10 epochs, except Flex-DiffLM, which is trained for 15 epochs; training with the same single-GPU setup is still ongoing. The single- versus multi-GPU comparison is discussed in Appendix G.4. We produce all results using our own experimental setup.

C.2.1 STANDARD LANGUAGE MODEL

The autoregressive LM is a LLaMA-style decoder-only Transformer (Touvron et al., 2023; Vaswani et al., 2017) with 24 layers, a hidden dimension of 1024, 8 attention heads and rotary positional embeddings (Su et al., 2023). The model was trained for 5 epochs with a batch size of 20,000 tokens. Further training did not improve performance.

C.2.2 DENOISING LANGUAGE MODEL

The denoising LM is an encoder-decoder Transformer with 18 encoder layers, 6 decoder layers, 8 attention heads, a hidden dimension of 1024 and rotary positional embeddings. The model was trained for 10 epochs with a batch size of 20,000 tokens.

C.2.3 DIFFUSION LANGUAGE MODELS

All diffusion language models use a Diffusion Transformer (DiT) with 24 layers, a hidden dimension of 1024, 16 attention heads and rotary positional embeddings. The models use bidirectional self-attention and a dropout rate of 0.1 (Srivastava et al., 2014). USDM and Flex-DiffLM use time conditioning with an embedding dimension of 128. MDLM, ASR-DiffLM and Flex ASR-DiffLM do not use time conditioning. We didn't see a difference in training with time conditioning or without for MDLM and USDM. Flex-DiffLM and Flex ASR-DiffLM use batches of 10,000 non-blank tokens per GPU, with approximately 10,000 additional blank positions in the interleaved canvas. The other models use batches of 20,000 tokens. Flex-DiffLM additionally uses packed tensors, CUDA graphs, bfloat16, and parameter averaging every 100 updates across four GPUs. We observe better error-correction performance when Flex-DiffLM is trained with substitution-only corruption on 50% of the training examples and full flexible-length corruption on the remaining 50%. Count-head details are in Appendix A.3.

C.3 CODE-INFILLING MODELS

For code infilling, we adapt the training pipeline of Peng et al. (2025) by continuing to train Qwen2.5-Coder-0.5B (Hui et al., 2024) with a diffusion objective for 5 epochs. We adopt the evaluation code from DreamOn (Wu et al., 2026). Sampling and evaluation details are in Appendix F.

C.4 TTS AND ASR MODELS

We use Glow-TTS (Kim et al., 2020) as a text-to-speech model to generate audio data from text. For automatic speech recognition, we use a Conformer encoder with 16 layers, a hidden dimension of 1024 and approximately 420M parameters, trained with a CTC head (Graves et al., 2006). The CTC model is trained on the LibriSpeech ASR training data ($\sim$960 hours) for 100 epochs. We train two different ASR models: one with TTS data and one without it. For more details on training and inference, see Koch et al. (2025), whose setup we follow. In the forward process, we use their low data augmentation configuration. Despite this low setting, the pipeline still produces many errors, with the TTS-CTC forward process yielding a word error rate of approximately 20%.

D DECODING METHODS

This section provides the details of the decoding methods in Section 5. Throughout this section, x_1^T denotes the audio, a_1^S a candidate token sequence and $p_{\theta,s}$ the model's predicted clean-token distribution at position s.

D.1 RESCORING

For rescoring, a CTC model (Graves et al., 2006) first produces an N-best list through time-synchronous beam search (see Prabhavalkar et al. (2023) for details):

$$\mathcal{N}_{\mathrm{CTC}} = N\text{-best}\left[p_{\mathrm{CTC}}(\cdot \mid x_1^T)\right].$$

The hypotheses are then ranked by a combination of acoustic and language-model scores:

$$F_{\text{comb}}(a_1^S) = \lambda_{\text{CTC}} \log p_{\text{CTC}}(a_1^S \mid x_1^T) + \lambda_{\text{DiffLM}} F_{\text{DiffLM}}(a_1^S) - \lambda_{\text{prior}} \log p_{\text{prior}}(a_1^S). \quad (4)$$

Here, F_{DiffLM} is the diffusion language-model score. The sequence prior is $p_{\text{prior}}(a_1^S) = \prod_{s=1}^{S} p_{\text{prior}}(a_s)$. The label prior is estimated by averaging CTC posteriors over training frames, removing the blank symbol and renormalizing (Koch et al., 2025; Manohar et al., 2015). This term approximately compensates for the label preferences already captured by the CTC model. The final output is

$$\overset{*}{a}{}_1^{\overset{*}{S}} = \arg\max_{a_1^S \in \mathcal{N}_{\text{CTC}}} F_{\text{comb}}(a_1^S).$$

Ideally, we want to use the sequence log-likelihood as the language-model score. For a standard autoregressive language model, this score can be computed directly as

$$F_{\text{LM}}(a_1^S) = \log p_\theta(a_1^S) = \sum_{s=1}^{S} \log p_\theta(a_s \mid a_1^{s-1}).$$

For diffusion language models, exact sequence-likelihood evaluation is generally intractable (Austin et al., 2021; Sahoo et al., 2024). We therefore use an approximation.

D.1.1 Monte Carlo Estimation

For each hypothesis, a noise level τ and K corrupted sequences are sampled:

$$z_{\tau,k,1}^S \sim p_{\text{fwd}}(\cdot \mid a_1^S, \tau), \qquad k = 1, \ldots, K.$$

Here, k indexes Monte Carlo samples. The model predicts the clean tokens from each corrupted sequence. Naveriani et al. (2026) proposed different scoring functions for aggregating these predictions and evaluated them under different parameter settings. However, Monte Carlo approximation requires K model evaluations to calculate a score for a single hypothesis. A standard autoregressive language model can do this in a single forward pass.

D.1.2 Pseudo-Log-Likelihood Scoring

To reduce this cost, we propose a pseudo-log-likelihood score that evaluates each hypothesis in a single forward pass, without corrupting the input:

$$F_{\text{DiffLM-PLL}}(a_1^S) = \sum_{s=1}^{S} \log p_{\theta,s}\left(a_s \mid z_{0,1}^S = a_1^S, \tau = 0\right).$$

For MDLM, this score uses the network's clean-token probabilities without applying the carry-over rule. We substitute $F_{\text{DiffLM-PLL}}$ for F_{DiffLM} in Equation (4). For the rescoring stages below, we use this score for both original and corrected hypotheses. For Flex-DiffLM, the model input is the interleaved canvas of a_1^S at $\boldsymbol{\tau} = \mathbf{0}$; the ASR-aware models omit noise-level conditioning.

D.2 Joint Decoding

The joint decoding method of Naveriani et al. (2026) initializes denoising with a CTC hypothesis and uses acoustic information during each decoding step. The initial hypothesis is obtained by CTC greedy decoding:

$$\hat{y}_t = \arg\max_{v \in V_{\text{blank}}} p_{\text{CTC},t}(v \mid x_1^T), \qquad \hat{a}_1^{\hat{S}} = B(\hat{y}_1^T),$$

where B collapses consecutive repeated labels and removes blanks. The reverse process is then initialized at noise level τ_{start} with $z_{\tau_{\text{start}},1}^{\hat{S}} = \hat{a}_1^{\hat{S}}$.

In this work, we use one-step greedy joint decoding. For fixed-length models, we combine the labelwise CTC distribution, the clean-token prediction and the label prior as follows:

$$F_{\text{comb},s}(v) = \lambda_{\text{CTC}} \log p_{\text{CTC},s}^{\text{indep}}(v \mid x_1^T) + \lambda_{\text{DiffLM}} \log p_{\theta,s}\left(v \mid z_{\tau_{\text{start}},1}^{\hat{S}}, \tau_{\text{start}}\right) - \lambda_{\text{prior}} \log p_{\text{prior}}(v).$$

Here, $p^{\text{indep}}_{\text{CTC},s}$ is the labelwise CTC distribution defined below, and we set $\lambda_{\text{CTC}} = 1$. The final hypothesis is

$$\overset{*}{a}{}_1^{\hat{S}} = \arg\max_{a_1^{\hat{S}} \in V^{\hat{S}}} \sum_{s=1}^{\hat{S}} F_{\text{comb},s}(a_s).$$

In the case of Flex-DiffLM and Flex ASR-DiffLM, we perform joint decoding directly on a canvas containing blank positions and remove blanks afterwards, allowing hypothesis length to change.

D.2.1 Labelwise CTC Distribution

In our implementation, we obtain $p^{\text{indep}}_{\text{CTC},s}$ by forming maximal groups of consecutive frames with the same most probable CTC label $\hat{y}_t$ and discarding groups corresponding to the blank symbol. Identical labels separated by a blank remain separate groups. Let $\mathcal{I}_s$ denote the group of frames corresponding to label $\hat{a}_s$. We average their posterior distributions:

$$\bar{p}_{\text{CTC},s}(v \mid x_1^T) = \frac{1}{|\mathcal{I}_s|} \sum_{t \in \mathcal{I}_s} p_{\text{CTC},t}(v \mid x_1^T).$$

We then remove the blank symbol and renormalize over V:

$$p^{\text{indep}}_{\text{CTC},s}(v \mid x_1^T) = \frac{\bar{p}_{\text{CTC},s}(v \mid x_1^T)}{\sum_{u \in V} \bar{p}_{\text{CTC},s}(u \mid x_1^T)}, \qquad v \in V.$$

For variable-length diffusion models, we do not remove groups with blanks and do not do renormalization.

D.3 DiffLM-Sum

We extend joint decoding by combining diffusion predictions from several CTC hypotheses. For hypotheses of the same length, the predictions are weighted by their CTC scores:

$$p_{\text{sum},s}(a_s \mid x_1^T; \theta) = \frac{\sum_{\tilde{a}_1^{\tilde{S}} \in \mathcal{N}_{\text{CTC}}} p_{\text{CTC}}(\tilde{a}_1^{\tilde{S}} \mid x_1^T) p_{\theta,s}\left(a_s \mid \tilde{a}_1^{\tilde{S}}, \tau_{\text{start}}\right)}{\sum_{\tilde{a}_1^{\tilde{S}} \in \mathcal{N}_{\text{CTC}}} p_{\text{CTC}}(\tilde{a}_1^{\tilde{S}} \mid x_1^T)}.$$

We replace $\log p_{\theta,s}$ with $\log p_{\text{sum},s}$ in joint decoding and use label-synchronous beam search over $F_{\text{comb},s}$ to obtain the candidate set $\mathcal{M}_{\text{comb}}$. We then rescore the union $\mathcal{H} = \mathcal{N}_{\text{CTC}} \cup \mathcal{M}_{\text{comb}}$ using

$$\begin{aligned} F_{\text{final}}(a_1^S) = \log p_{\text{CTC}}(a_1^S \mid x_1^T) + \lambda_{\text{sum}} F_{\text{DiffLM-Sum}}(a_1^S) \\ + \lambda_{\text{PLL}} F_{\text{DiffLM-PLL}}(a_1^S) - \lambda_{\text{prior}} \log p_{\text{prior}}(a_1^S), \end{aligned}$$

where

$$F_{\text{DiffLM-Sum}}(a_1^S) = \sum_{s=1}^{S} \log p_{\text{sum},s}(a_s \mid x_1^T; \theta).$$

D.4 Soft-Input Decoding

Instead of feeding hard labels to the diffusion model, we multiply the CTC posterior distributions by the model's embedding matrix and feed the resulting embeddings to the model. The posterior distribution at position s is

$$\pi_s(v) = p^{\text{indep}}_{\text{CTC},s}(v \mid x_1^T), \qquad v \in V.$$

The main motivation is to use as much information from CTC as possible in a single forward pass.

D.5 Denoising with Rescoring

We apply joint decoding independently to every hypothesis in the CTC N-best list:

$$\mathcal{D} = \left\{ \text{JD}(\tilde{a}_1^{\tilde{S}}) \ : \ \tilde{a}_1^{\tilde{S}} \in \mathcal{N}_{\text{CTC}} \right\},$$

where JD represents the joint decoding algorithm with $\tilde{a}_1^{\tilde{S}}$ as the CTC hypothesis. We merge the original hypotheses with denoised sequences $\mathcal{H} = \mathcal{N}_{\text{CTC}} \cup \mathcal{D}$ and do rescoring.

D.6 Configuration

- Standard LM (first-pass search): beam size 64.
- Denoising LM (DLM-Sum): CTC beam size 20, output beam size 12.
- Rescoring: CTC beam size 64.
- Joint Decoding: greedy CTC initialization, one correction step, greedy output.
- DiffLM-Sum: CTC beam size 16, output beam size 64, concatenation with the 16 original CTC hypotheses, deduplication before rescoring.
- Soft-Input Decoding: CTC posterior input, one correction step, greedy output.
- Denoising with Rescoring: CTC beam size 64, one correction step per hypothesis, concatenation of original hypotheses, deduplication before rescoring.
- USDM noise levels: $\tau_{\text{start}} = 0.1$ for Joint Decoding, DiffLM-Sum and Soft-Input Decoding, $\tau_{\text{start}} = 0.3$ for Denoising with Rescoring, $\tau = 0$ for final hypothesis scoring. Numbers were manually tuned.
- Other diffusion models: $\boldsymbol{\tau} = \mathbf{0}$ for Flex-DiffLM recognition, no time conditioning for the ASR-aware models.
- Score weights: CTC weight fixed to 1, remaining weights tuned on dev-other and reused across evaluation sets.

The runtime measurement protocol is described in Appendix E.2.7.

E Generation and Evaluation

E.1 Sampling Settings

We generate 1,500 unconditional samples and 1,500 continuations per configuration. We use nucleus sampling with $p = 0.9$, temperature 1.0 for unconditional generation, and temperature 0.6 for continuation. Diffusion sampling uses ancestral transitions with the substitution ψ-correction ($\psi = 0.05$).

- MDLM and USDM use 96 backbone evaluations and 16 visible output tokens.
- Fixed-length Flex-DiffLM uses 96 substitution updates with $\tau^{\text{len}} = 0$ and 16 visible output tokens.
- Variable-length Flex-DiffLM repeats a length update followed by two substitution updates 32 times, giving 96 backbone evaluations. Before this schedule, two initialization updates insert one token into every editable gap. Starting from an empty editable sequence, these updates produce one and then three tokens. The ordinary schedule then starts at $\boldsymbol{\tau} = (1, 1)$, giving 98 backbone evaluations in total.
- Fixed-length AR generation suppresses EOS and stops after 16 generated tokens. Variable-length AR generation stops at EOS, with a safety limit of 2,048 generated tokens.

All noise levels decrease linearly along their respective update grids. The final substitution update uses argmax. In continuation, the prefix remains fixed and only the suffix and its boundary gaps are editable.

E.2 Evaluation Metrics

E.2.1 Perplexity

Perplexity (PPL) measures the degree of uncertainty of a language model when predicting the ground-truth sequence. For a dataset $\mathcal{D}$ containing N_{tok} scored tokens, it is the exponential of the average negative log-likelihood:

$$\text{PPL}(\mathcal{D}; p_\theta) = \exp\left(-\frac{1}{N_{\text{tok}}} \sum_{a_1^S \in \mathcal{D}} \log p_\theta(a_1^S) \right). \tag{5}$$

Lower PPL is better.

E.2.2 GENERATIVE PERPLEXITY

One of the main metrics for comparing the text generation capabilities of language models is perplexity (PPL). Unfortunately, computing the exact PPL of diffusion language models is intractable. Therefore, an external autoregressive model, such as GPT-2 Large, is used to compute the perplexity of sequences generated by DiffLMs. This metric is called generative perplexity (Gen-PPL):

$$\text{Gen-PPL} = \exp\left(-\frac{1}{N_{\text{tok}}}\sum_{i=1}^{N_{\text{tok}}}\log p_{\text{AR}}(w_i \mid c_i)\right).$$

Here, N_{tok} is the total number of scored tokens, w_i is a scored token from the evaluator's tokenizer and c_i is the available context.

We use GPT-2 Large and its tokenizer as the evaluator. Unconditional samples are concatenated in example order with EOS separators and divided into non-overlapping 1,024-token blocks. Separator targets are excluded from the loss but remain in the context. The first token of each block is not scored. We aggregate token-level negative log-likelihoods before exponentiation rather than averaging sample PPLs. The final incomplete block is included if it contains at least two tokens. Empty outputs contribute only an unscored EOS separator.

E.2.3 CONTINUATION PERPLEXITY

Continuation PPL evaluates generated suffixes conditioned on fixed prefixes. We tokenize each prefix and continuation together, separated by a space, using GPT-2 Large's tokenizer. Only continuation tokens contribute to the loss; prefix tokens provide context. Examples are scored separately, without concatenation across sentences or an appended EOS token.

E.2.4 ENTROPY

We compute the unigram entropy of each generated sequence and average it over all samples. For a sequence a_1^S, the empirical token distribution is

$$\hat{p}(v \mid a_1^S) = \frac{1}{S}\sum_{s=1}^{S}\mathbf{1}_{\{a_s=v\}}, \qquad v \in V,$$

where $\mathbf{1}_{\{a_s=v\}}$ is a function that returns 1 if $a_s = v$ and 0 otherwise.

For N_{seq} generated sequences $a_{i,1}^{S_i}$, the average entropy is

$$\text{Entropy} = -\frac{1}{N_{\text{seq}}}\sum_{i=1}^{N_{\text{seq}}}\sum_{v\in V}\hat{p}(v \mid a_{i,1}^{S_i})\log\hat{p}(v \mid a_{i,1}^{S_i}).$$

E.2.5 MAUVE

MAUVE was proposed by Pillutla et al. (2021) and compares text distributions using KL divergence computed from their vector representations. In our case, these representations are extracted from GPT-2 Large. Specifically, we extract the hidden state of the last token from the final Transformer layer, after the final layer normalization and before the output projection. We use this vector as the representation of the whole sequence. We then group the representations from both sets into K clusters using k-means clustering. Let $p_{\text{gen}}(k)$ and $p_{\text{ref}}(k)$ denote the proportions of generated and reference sequences assigned to cluster k, respectively. We define a mixture of these distributions:

$$p_\lambda(k) = \lambda p_{\text{gen}}(k) + (1-\lambda)p_{\text{ref}}(k), \qquad \lambda \in (0,1).$$

For each value of λ, we compute

$$x_\lambda = \exp(-c\, D_{\text{KL}}(p_{\text{ref}} \| p_\lambda)),$$
$$y_\lambda = \exp(-c\, D_{\text{KL}}(p_{\text{gen}} \| p_\lambda)),$$

where $c > 0$ is a fixed scaling constant. These points form a curve, and MAUVE is the area under it:

$$\text{MAUVE} = \text{AUC}\left(\{(x_\lambda, y_\lambda) : \lambda \in (0,1)\}\right).$$

Here, AUC stands for area under the curve formed by the points (x_λ, y_λ) as λ varies. MAUVE ranges from 0 to 1. Higher values indicate closer agreement between the generated and reference text distributions.

We use $c = 5$ and determine the number of clusters automatically:

$$K = \max\left(2, \text{round}\left(\frac{\min(N_{\text{gen}}, N_{\text{ref}})}{10}\right)\right),$$

where N_{gen} and N_{ref} are the numbers of generated and reference texts, respectively. With 1,500 nonempty texts in each set, this gives $K = 150$.

E.2.6 Word Error Rate

The main evaluation metric for speech recognition is Word Error Rate (WER), which denotes how much the text recognized by the ASR system differs from the ground truth. It is computed based on the Levenshtein edit distance between the recognized transcription and the reference transcription:

$$\text{WER} = \frac{\text{Substitutions} + \text{Deletions} + \text{Insertions}}{\text{Number of words in the reference}}. \tag{6}$$

WER is reported as a percentage by multiplying this ratio by 100. Lower WER is better. Corpus WER is computed by summing the edit counts and reference lengths over all utterances before taking the ratio.

E.2.7 Real-Time Factor

Real-time factor (RTF) measures decoding time relative to audio duration:

$$\text{RTF} = \frac{T_{\text{decoding}}}{T_{\text{audio}}}. \tag{7}$$

Lower RTF is better. An RTF below one means that the measured processing is faster than the duration of the audio.

For dev-other, the timing recipe uses the approximate audio duration of 5.3 hours (19,080 seconds) consistently across systems. Each forward stage is measured three times using the elapsed time reported by RETURNN, and the arithmetic mean is used. For methods with several stages, their mean runtimes are added.

F Code-Infilling Evaluation

We evaluate HumanEval single-line and multi-line infilling and the SantaCoder infilling benchmark using DreamOn's evaluation code (Wu et al., 2026). The code models are initialized from Qwen2.5-Coder-0.5B and trained as described in Appendix C.

Sampling settings. The configurations use temperature 0.6, nucleus threshold 0.9, and $\psi = 0.05$ for the ancestral diffusion sampler. Fixed-length models use 96 backbone evaluations. Variable-length Flex-DiffLM alternates 32 length updates with 64 substitution updates and starts from an empty editable span. The prefix and suffix remain fixed. The final substitution step is greedy. For DreamOn, we follow the authors' sampling strategy without modification.

Length information. For fixed-length models, we evaluate the predefined length settings reported in Table 1. Oracle-length runs receive the reference length when constructing the initial canvas. Variable-length Oracle instead starts from an empty span and uses reference-derived insertion counts at each gap.

G Additional Experiments

G.1 Initialization

We compare zero, one, two, and three forced insertion updates before the ordinary sampling schedule. Starting from an empty sequence, these updates produce 0, 1, 3, and 7 initial tokens, respectively.

Table 6: Initialization ablation for variable-length Flex-DiffLM.

Initial updates	Total steps	Gen. PPL ↓	Mean length	Entropy ↑	MAUVE ↑	Cont. PPL ↓
0	96	195.4	6.0	1.65	0.46	23.2
1	97	94.4	8.5	1.98	0.60	18.0
2	98	40.8	14.7	2.49	0.84	16.0
3	99	30.6	29.0	3.09	0.84	14.4

Without initialization, unconditional outputs average 6.0 tokens. Two initial updates increase the mean to 14.7 tokens and improve Gen PPL, entropy, and MAUVE. This is the configuration used in the main comparison. Three updates generate substantially longer sequences, illustrating that initialization changes the output-length distribution as well as quality.

G.2 Fewer Sampling Steps

Table 7: Text generation with fewer sampling steps and the same checkpoints as Table 2. Variable-length Flex-DiffLM uses 16 substitution, eight length, and two initialization updates. Each configuration uses 1,500 unconditional samples and 1,500 continuations.

Model / length	Total steps	Gen. PPL ↓	Mean length	Entropy ↑	MAUVE ↑	Cont. PPL ↓
MDLM	16	52.5	16.0	2.67	0.92	18.1
USDM	16	40.6	16.0	2.65	0.89	18.9
Flex-DiffLM, fixed	16	44.2	16.0	2.66	0.89	18.7
Flex-DiffLM, variable	26	47.1	13.6	2.44	0.78	16.8

The experiments with fewer sampling steps use 16 backbone evaluations for MDLM, USDM, and fixed-length Flex-DiffLM. Variable-length Flex-DiffLM uses 16 substitution updates, eight length updates, and two initialization updates, for 26 evaluations. The backbone checkpoints and evaluation protocol are shared with the main generation comparison. With fewer sampling steps, the fixed-length models retain MAUVE close to their 96-step values, while continuation PPL increases. Variable-length Flex-DiffLM reaches MAUVE 0.78 with 26 evaluations, compared with 0.84 using 98 evaluations.

G.3 Longer ASR-Aware Training

Table 8: Flex ASR-DiffLM after 10 and 25 full training epochs on one GPU. Both systems combine a CTC acoustic model trained with TTS data with DiffLM-Sum, 16 CTC hypotheses, and output beam size 64.

Epochs	WER [%] ↓			
	dev-clean	dev-other	test-clean	test-other
10	1.50	3.41	1.77	3.77
25	1.52	3.36	1.72	3.69

We compare Flex ASR-DiffLM after 10 and 25 full training epochs on one GPU. Both use the CTC acoustic model trained with TTS data and DiffLM-Sum with 16 CTC hypotheses, output beam size 64, and one correction step. The original CTC hypotheses are included before final selection, which uses the ASR-aware model's own scoring function. The 25-epoch model reaches 3.36% WER on dev-other and 3.69% on test-other. These results extend the main ten-epoch comparison without introducing an additional autoregressive rescoring model.

G.4 Single- and Multi-GPU Training

For Flex ASR-DiffLM, we compare single-GPU training with the same multi-GPU training setup used for Flex-DiffLM. As shown in Table 9, single-GPU training yields better results.

Table 9: Single- and multi-GPU training of Flex ASR-DiffLM, evaluated with the TTS-trained CTC model and DiffLM-Sum using 16 CTC hypotheses and output beam size 64.

GPUs	Epochs	WER [%] ↓			
		dev-clean	dev-other	test-clean	test-other
1	10	1.50	3.41	1.77	3.77
1	25	1.52	3.36	1.72	3.69
4	25	1.54	3.45	1.79	3.83

G.5 Subword Segmentation Mismatch

We noticed during the investigation that a CTC hypothesis and the canonically tokenized reference can represent the same word with different subword segmentations, as shown in Table 10. These differences introduce artificial token-level errors. Identifying and resolving this mismatch was a crucial finding that led to a significant improvement in error-correction performance. We first align the hypothesis and reference at the word level, then use the CTC segmentation for words that were recognized correctly. Token-level alignment is performed after this normalization.

Table 10: Examples of subword tokenization mismatches between a CTC hypothesis and the canonically tokenized reference.

Word	CTC hypothesis segmentation	Reference segmentation
GABLES	_GA BLE S	_G ABLE S
FURTHEST	_FUR THE ST	_FUR TH EST

Table 11 shows the effect of this correction on Flex ASR-DiffLM rescoring after five training epochs.

Table 11: Effect of fixing the subword segmentation mismatch on Flex ASR-DiffLM rescoring after five training epochs.

Flex ASR-DiffLM training setup	Decoding method	WER [%] on dev-other
With mismatch	rescoring	4.84
Without mismatch	rescoring	4.38

H Generated Samples

We show the first two unconditional records in the registered evaluation files for each configuration in Table 2. Text is reproduced without manual correction or selection by quality. The sample indices identify records in the shared evaluation ordering.

Real text (dev-clean) *Sample 0.* MISTER QUILTER IS THE APOSTLE OF THE MIDDLE CLASSES AND WE ARE GLAD TO WELCOME HIS GOSPEL

Sample 4. LINNELL'S PICTURES ARE A SORT OF UP GUARDS AND AT EM PAINTINGS AND MASON'S EXQUISITE IDYLLS ARE AS NATIONAL AS A JINGO POEM MISTER BIRKET FOSTER'S LANDSCAPES SMILE AT ONE MUCH IN THE SAME WAY THAT MISTER CARKER USED TO FLASH HIS TEETH AND MISTER JOHN COLLIER GIVES HIS SITTER A CHEERFUL SLAP ON THE BACK BEFORE HE SAYS LIKE A SHAMPOOER IN A TURKISH BATH NEXT MAN

AR, fixed length *Sample 0.* I AM DOING MY BEST FOR THE OWNER'S SON HE IS ONE OF THE

Sample 4. THE LAD STOOPED FORWARD TO PUSH HIS WAY THROUGH THE PEOPLE AND THE MERCHANT DREW NEAR

AR, variable length *Sample 0.* I AM DOING MY BEST FOR THE OWNER AND I AM GLAD HE WILL HAVE THE LITTLE BED PUT INTO MY TENT HERE

Sample 4. THE LAD FACED A HOST OF DIFFICULTIES

MDLM, 96 steps *Sample 0.* BUT THEY MAINTAINED AN IMPLICIT OBEDIENCE TO THE IMPERATIVE DUTIES ENTAILING UPON THEIR RELIGION

Sample 4. YOU WILL FIND IT IN THE FIRST MONTHS OF THE SPRING WHICH WILL BE VERY CONVENIENT

USDM, 96 steps *Sample 0.* SHE TOLD THE DOCTOR WHO WAS IN A VERY HEALTHY CONDITION THAT THE TREES WERE GOOD

Sample 4. THIS TENDS TO ENABLE THE ROYALISTS WITH THEIR KNOWLEDGE OF THE LANGUAGE TO READ

Flex-DiffLM, fixed length, 96 steps *Sample 0.* THE CROWD ADVANCING ALONG THE STREET WAS IN A VERY HEALTHY CONDITION AMONGST THEM ALL

Sample 4. HIS VARIOUS PRECAUTIONS AGAINST DOUBT ARE SUFFICIENTLY OBVIOUS FROM HIS KNOWLEDGE OF THE SCRIPTURES

Flex-DiffLM, variable length, 96+2 steps *Sample 0.* I HAVE LEARNED SINCE THAT HE MADE NO CHARGE AGAINST ME BUT THAT I NEVER SAID ANYTHING TO DISCREDIT HIM

Sample 4. THEY KILL CARPPANS WHEN NOT FLYING