5. Diffusion Models in Language

We have seen that image generation can be treated as the evolution of a state along a differential equation trajectory. The resulting diffusion models provide a powerful tool to describe the bidirectional transformation between prior and data distributions. In this section, we extend this approach to the generation of token sequences, which is a fundamental problem in modern **natural language processing** (NLP).

A key challenge is that tokens are discrete variables that carry linguistic meaning. Specifically, unlike the continuous space of images, there is no meaningful intermediate state between two discrete tokens. Moreover, the categorical and sequential nature of language means that even small changes in token identity or order can drastically alter linguistic meaning. Therefore, we need to adapt diffusion models to token sequence modeling by modifying existing methods and algorithms. In this section, we begin by introducing autoregressive and non-autoregressive generation paradigms. We then consider two approaches to diffusion modeling in language: embedding-space diffusion and discrete diffusion. The former is based on continuous-time diffusion models and operates in the embedding space of tokens. The latter directly defines generation in the discrete token space and generally leverages token masking and prediction mechanisms. Furthermore, we discuss the application of diffusion Transformer architectures and flow matching in token sequence generation, as well as the issues of inference and generation control.

Diffusion modeling is a wide-ranging topic. Here, we try to discuss it from the perspective of differential equations, and so connect key concepts to those presented in previous sections. We also present methods that are not originally motivated by differential equations but are included to have a comprehensive discussion.

5.1 Token Sequence Generation

We first present the concepts of **autoregressive (AR) generation** and **non-autoregressive (NAR) generation**, and then show that diffusion modeling can be treated as an NAR generation problem.

5.1.1 Autoregressive Generation vs. Non-autoregressive Generation

Token sequence generation (or **sequence generation** for short) is a fundamental task in NLP, and lays the foundation for **large language models** (LLMs) [Radford et al., 2018]. The dominant approach is AR generation[1]. In NLP, a typical implementation is left-to-right generation, which predicts one token at a time given all its preceding tokens. Formally, let $\mathbf{x} = \{x_1, ..., x_n\}$ be a

1. In statistics, *regression* refers to estimating the relationship between a dependent variable (Y) and independent variables (X). The prefix *auto* means *self* in Greek. Therefore, *autoregression* means performing a regression analysis where the independent variables are the lagged values of the variable itself. Instead of predicting Y based on X, one predicts Y_{today} based on $Y_{\text{yesterday}}$.

token sequence. The probability of this sequence is defined using the chain rule of probability

$$\begin{aligned} \Pr(\mathbf{x}) &= \Pr(x_1, ..., x_n) \\ &= \prod_{i=1}^{n} \Pr(x_i | x_{<i}). \end{aligned} \tag{1}$$

Here, x_i represents the token being generated at the current step, and $x_{<i}$ represents all the tokens generated previously. $\Pr(x_i|x_{<i})$ is the conditional probability of the next token, which reflects the nature of left-to-right generation. There are many ways to implement $\Pr(x_i|x_{<i})$. A popular method is to use neural networks to model this probability. For example, LSTMs and Transformers are both widely used architectures in neural language modeling.

In many text generation tasks, the generation is conditioned on user-provided information. We denote this contextual information as $\mathbf{c}$, which can be broadly viewed as a sequence of variables $\mathbf{c} = \{c_1, ..., c_m\}$. Then, the generation task is to model the probability of predicting the following token sequence $\mathbf{x}$ given the context $\mathbf{c}$

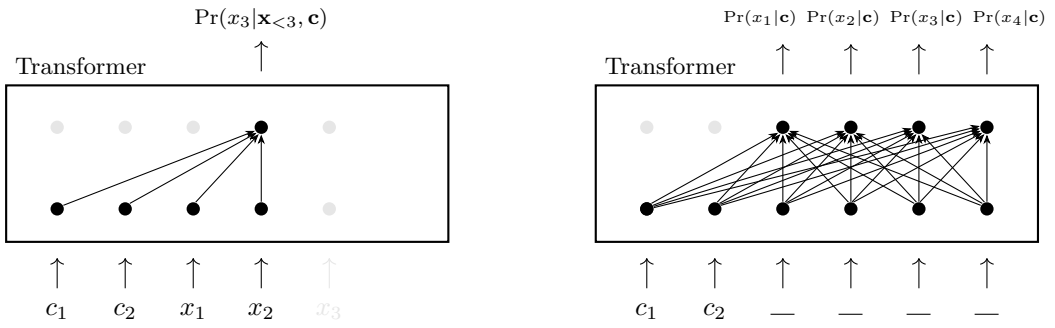
$$\Pr(\mathbf{x}|\mathbf{c}) = \prod_{i=1}^{n} \Pr(x_i | x_{<i}, \mathbf{c}). \tag{2}$$

Here $\mathbf{c}$ can be either a discrete token sequence (e.g., a prompt) or a real-valued vector sequence (e.g., representations generated by a neural network). In general, $\mathbf{c}$ can be defined according to the task of interest. For example, we can treat it as a source sentence in translation, or a prompt in QA, or a real-valued representation of history in an external memory.

The primary advantage of AR generation lies in its simplicity. This leads to several desirable properties. For example, it aligns well with how humans perceive speech and writing, and is intuitive for tasks like dialogue and storytelling. Moreover, AR generation can naturally handle sequences of varying lengths. The model continues generating until it outputs a special End-of-Sequence (EOS) token, rather than being forced to fill a fixed-size template. However, AR generation has disadvantages that have long been concerns in the NLP community. A major disadvantage is its slow generation speed. Because step i depends on step $i-1$, generation cannot be parallelized. To generate a sentence with 100 tokens, the model must run the forward pass 100 times sequentially. This makes real-time applications computationally expensive. Another disadvantage lies in its unidirectional context scope. When predicting token x_i, the model can only see to the left ($x_{<i}$). It cannot look ahead to the right to see how the sentence should end.

NAR generation is an alternative to AR generation that aims to break the sequential dependency chain [Gu et al., 2018]. Instead of generating tokens one by one, NAR models generate all tokens in the sequence in parallel. Mathematically, this approach relies on the assumption of conditional independence among target tokens given the input. The probability of the sequence $\mathbf{x}$ can then be factorized as

$$\Pr(\mathbf{x}|\mathbf{c}) = \prod_{i=1}^{n} \Pr(x_i|\mathbf{c}) \tag{3}$$



(a) Autoregressive Generation (Single Step)

(b) Non-autoregressive Generation (Single Step)

Figure 1: Autoregressive (AR) vs. non-autoregressive (NAR) generation. $\mathbf{c} = \{c_1, c_2\}$ denotes the user-provided context, and $\mathbf{x} = \{x_1, x_2, x_3, x_4\}$ denotes the target sequence. We aim to model the probability of $\mathbf{x}$ given $\mathbf{c}$. (a) In AR generation, tokens are predicted sequentially, and the prediction of x_i depends on the history $\mathbf{x}_{<i}$ and $\mathbf{c}$. (b) In NAR generation, all tokens in the sequence are predicted in parallel. The dashes in (b) represent a template of 4 mask tokens (given a target length).

In this formulation, the prediction of token x_i does not depend on other target tokens x_j ($j \neq i$). Thus, NAR generation is efficient for inference. By decoupling the dependencies between tokens, the model can predict the entire sequence in a single forward pass, which reduces the number of inference steps from $O(n)$ to $O(1)$. This parallelism allows modern GPUs to be utilized more efficiently, and thus makes NAR models attractive for real-time applications where low latency is critical. In addition, NAR models can make full use of information from both the past and the future positions during the generation process. Figure 1 illustrates the differences between AR and NAR generation.

It should be noted, however, that the conditional independence assumption also has limitations. For example, in natural language, there are often multiple valid ways to complete a sentence. Without knowing what tokens have been selected at other positions, an NAR model might mix different valid patterns, leading to the so-called multimodality problem[2].

5.1.2 Diffusion Modeling as Non-autoregressive Generation

Despite the above limitations, the parallel nature of NAR generation aligns perfectly with diffusion models, which typically operate on the entire data dimensionality simultaneously. Therefore, most of the diffusion models in NLP are based on the NAR generation paradigm. To understand this connection, it is helpful to generalize the concept of NAR generation beyond a single-step prediction. Let us reimagine the generation process as a dynamical system evolving over time t. We define $\mathbf{s}(t)$ as the intermediate state of the system at time t. This abstract state $\mathbf{s}(t)$ serves as a unified representation: it can be either a sequence of discrete variables (e.g., corrupted tokens) or

2. This problem may result in repetition, omission, or grammatical incoherence, e.g., choosing "New" at position i and "London" at position $i+1$ when the valid entities are "New York" or "London".

a sequence of continuous vectors (e.g., token embeddings). During generation, the system evolves backwards in time from $t = T$ to $t = 0$. The states of this system are defined as follows

- The starting point $\mathbf{s}(T)$. This represents a sample drawn from a prior distribution (e.g., Gaussian noise vectors or completely random tokens). It contains no semantic information.
- The intermediate states $\mathbf{s}(t)$ (for $0 < t < T$). These states represent the evolving latent variables. They can be viewed as noisy approximations of the target data.
- The end point $\mathbf{s}(0)$. The state $\mathbf{s}(0)$ represents the fully denoised signal. Finally, a deterministic or stochastic mapping (e.g., rounding or an argmax operation) projects $\mathbf{s}(0)$ back to the valid token sequence $\mathbf{x} = \{x_1, \ldots, x_n\}$.

In this process, we need to incorporate the condition $\mathbf{c}$ into each state $\mathbf{s}(t)$ to form a joint state $(\mathbf{s}(t), \mathbf{c})$. Hence the evolution can be described as

$$(\mathbf{s}(T), \mathbf{c}) \to \cdots \to (\mathbf{s}(t), \mathbf{c}) \to \cdots \to (\mathbf{s}(0), \mathbf{c}) \to \mathbf{x}. \tag{4}$$

The final map $(\mathbf{s}(0), \mathbf{c}) \to \mathbf{x}$ is generally performed by using a separate system (e.g., mapping embeddings to tokens). The steps $(\mathbf{s}(T), \mathbf{c}) \to \cdots \to (\mathbf{s}(t), \mathbf{c}) \to \cdots \to (\mathbf{s}(0), \mathbf{c})$ can be viewed as the evolution of a dynamical system, as in standard diffusion models. This process is illustrated in Figure 2.

In the remainder of this section, we use diffusion models as tools to implement NAR generation. We categorize the approaches based on how they define the state space of the dynamical system $\mathbf{s}(t)$. We first introduce embedding-space diffusion, which maps tokens to continuous vectors to apply continuous Gaussian diffusion. We then discuss discrete diffusion, which defines the corruption and denoising processes directly on the discrete tokens.

5.2 Diffusion in Continuous Space: Embedding-Space Diffusion

In this subsection, we introduce the embedding-space diffusion framework, which adapts continuous diffusion models to token sequence generation.

5.2.1 General Framework

One intuitive approach to applying diffusion models to text is to map discrete tokens into a continuous embedding space $\mathbb{R}^d$, so that the diffusion and generation processes can operate in a continuous domain. Several prominent works are based on this embedding-space diffusion framework [Li et al., 2022; Strudel et al., 2022; Gong et al., 2023]. In these methods, the first step is to transform each token of a sequence into a d-dimensional real-valued vector (called token embedding). Formally, let V be a vocabulary of size $|V|$. Each token $x \in V$ is typically represented as a one-hot vector and projected into a dense vector. Then, a sequence of tokens $\mathbf{x} = \{x_1, \ldots, x_n\}$ can be mapped to a sequence of continuous vectors via an embedding function

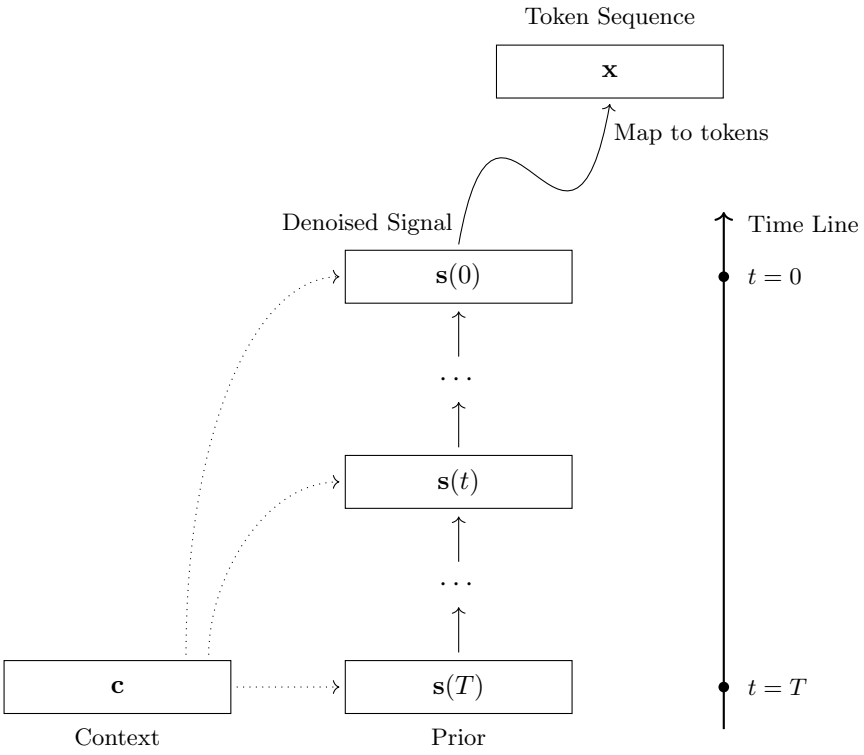


Figure 2: Schematic illustration of NAR generation based on diffusion models. The generation process is modeled as the evolution of a state $\mathbf{s}(t)$ over time t. Here $\mathbf{s}(T)$ denotes the starting point (prior), and $\mathbf{s}(0)$ denotes the end point (fully denoised signal). The contextual information $\mathbf{c}$ is injected into the state during generation. The final output is a token sequence that is mapped from $\mathbf{s}(0)$.

Embed$(\cdot)$, as follows

$$\begin{aligned} \text{Embed}(\mathbf{x}) &= [\mathbf{e}_1, \ldots, \mathbf{e}_n], \\ &= \mathbf{e} \end{aligned} \tag{5}$$

where $\mathbf{e}_i \in \mathbb{R}^d$ denotes the embedding at position i, and $\mathbf{e} \in \mathbb{R}^{d \times n}$ denotes the sequence of embeddings (stacked as a matrix). Likewise, we can transform the context token sequence $\mathbf{c}$ into an embedding sequence $\mathbf{e}_c \in \mathbb{R}^{d \times n_c}$.

A simple way to define Embed$(\cdot)$ is via a static embedding matrix $\mathbf{E} \in \mathbb{R}^{d \times |V|}$ [Mikolov et al., 2013a;b]. While simple, this approach treats tokens as independent units, and thus ignores their contextual dependencies in the sequence. Alternatively, one can map the entire token sequence into a sequence of contextualized representations. For instance, pre-trained LLMs can be used to encode both the tokens and their surrounding context [Devlin et al., 2019; Brown et al., 2020]. Note that this transformation is not limited to a one-to-one correspondence between tokens and vectors, nor is it constrained to a fixed sequence length n. Instead, we can employ higher-level

abstractions to map the token sequence into a latent continuous sequence of a different length n' (where n' need not equal n). For instance, by using VAEs, discrete sequences can be compressed into denser, semantically rich latent representations. In this latent diffusion paradigm [Rombach et al., 2022], the diffusion process operates on these compressed latent variables rather than on raw token embeddings.

Once the tokens are represented as embeddings, the generation problem can be formulated using a diffusion model in a standard manner. Formally, let $\mathbf{e}_c$ be the context embedding sequence. We define $\bar{\mathbf{s}}(t)$ as the augmented state combining $\mathbf{e}_c$ and $\mathbf{s}(t)$

$$\bar{\mathbf{s}}(t) \quad = \quad [\mathbf{e}_c, \mathbf{s}(t)]. \tag{6}$$

We treat the concatenation of $\mathbf{e}_c$ and $\mathbf{e}$ as the initial state $\bar{\mathbf{s}}(0) = [\mathbf{e}_c, \mathbf{e}] \in \mathbb{R}^{d\times(n_c+n)}$ of a continuous diffusion process. Then we define a forward process that gradually adds Gaussian noise to the target embeddings over time $t \in [0, T]]$, while keeping the context embeddings fixed. It transforms the meaningful token embeddings into standard Gaussian noise $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

As described previously, there are several models to choose from. For example, we can describe the diffusion process by an SDE

$$d\bar{\mathbf{s}}(t) \quad = \quad f(\bar{\mathbf{s}}(t), t)dt + g(t)d\mathbf{w}, \tag{7}$$

where $f(\bar{\mathbf{s}}(t), t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and $\mathbf{w}$ is a standard Wiener process. The generation process corresponds to the reverse-time SDE, which reconstructs $\bar{\mathbf{s}}(0)$ from a state $\bar{\mathbf{s}}(T)$ where the target sequence part is Gaussian noise:

$$d\bar{\mathbf{s}}(t) \quad = \quad [f(\bar{\mathbf{s}}(t), t) - g(t)^2\nabla_{\bar{\mathbf{s}}(t)} \log p_t(\bar{\mathbf{s}}(t))]dt + g(t)d\bar{\mathbf{w}}, \tag{8}$$

where $\bar{\mathbf{w}}$ is the reverse-time Wiener process, and $\nabla_{\bar{\mathbf{s}}(t)} \log p_t(\bar{\mathbf{s}}(t))$ is the score function. Alternatively, one can use flow matching to learn a vector field that pushes forward the distribution from Gaussian noise to the data distribution. A detailed discussion of these methods is omitted here. Readers can refer to Section 4 for more details on various diffusion models.

Note that since the context $\mathbf{c}$ and its embedding $\mathbf{e}_c$ remain constant throughout the process, we can simply use $\mathbf{s}(t)$ instead of $\bar{\mathbf{s}}(t)$ to describe the diffusion and generation processes. Hence Eqs. (7) and (8) can be equivalently rewritten as

$$d\mathbf{s}(t) \quad = \quad f(\mathbf{s}(t), t)dt + g(t)d\mathbf{w}, \tag{9}$$

$$d\mathbf{s}(t) \quad = \quad [f(\mathbf{s}(t), t) - g(t)^2\nabla_{\mathbf{s}(t)} \log p_t(\mathbf{s}(t))]dt + g(t)d\bar{\mathbf{w}}. \tag{10}$$

A key difference from computer vision models is that in NLP we typically employ Transformers as the backbone architecture for learning score functions or vector fields. Given a state $\bar{\mathbf{s}}(t) = [\mathbf{e}_c, \mathbf{s}(t)] \in \mathbb{R}^{d\times(n_c+n)}$, the Transformer outputs a real-valued matrix of the same shape, estimating the score or vector field at each position, as illustrated in Figure 3. Unlike image generation, where the spatial dimensions are typically fixed, natural language sequences are inherently variable in length. Thus we need to determine the target sequence length n during inference. Furthermore,

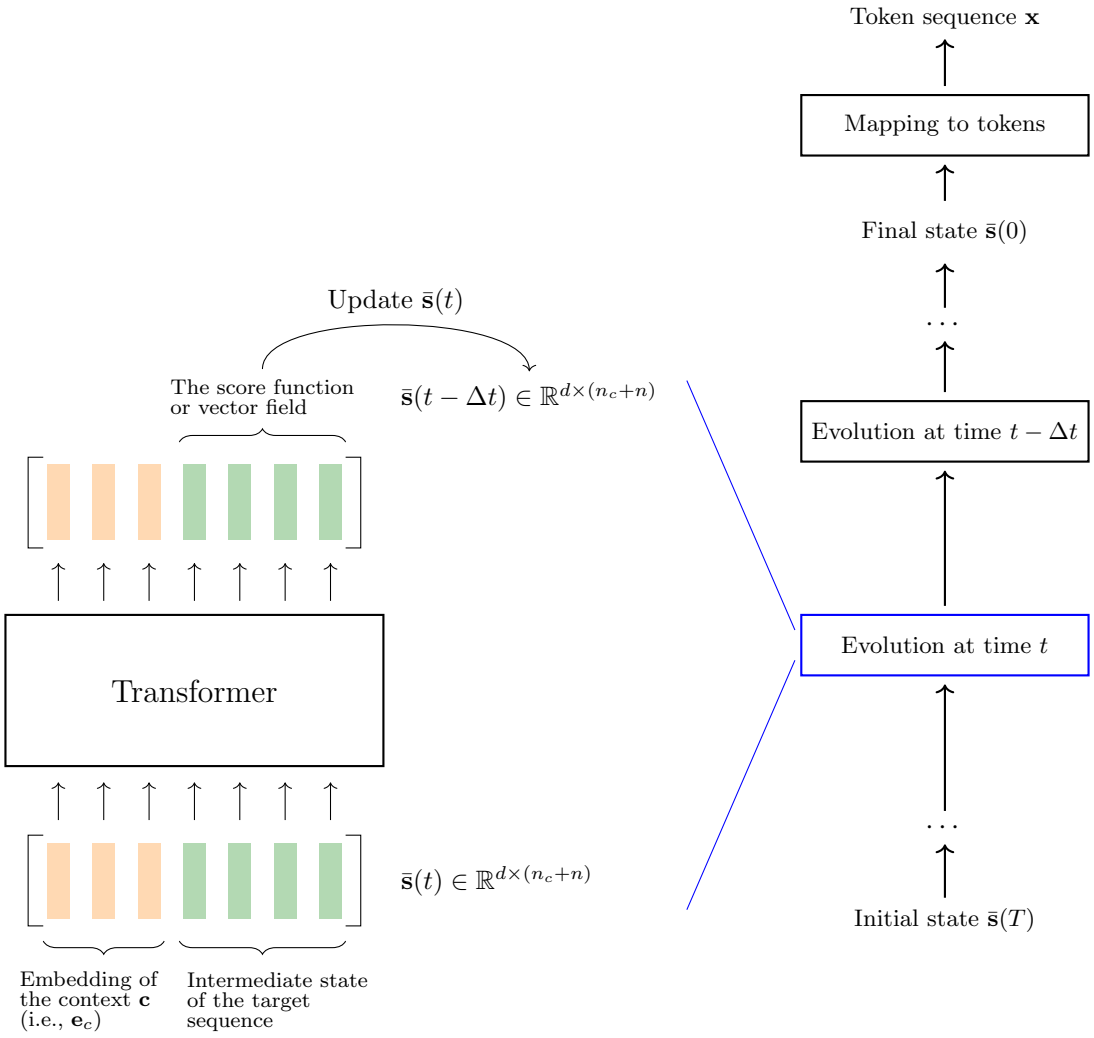


Figure 3: Schematic illustration of the Transformer-based generation process. The generation (i.e., reverse diffusion) process begins at $\bar{\mathbf{s}}(T)$, initialized as standard Gaussian noise for the target sequence. At each time step t, the state $\bar{\mathbf{s}}(t)$ represents the concatenation of the context embeddings and the noisy target embeddings. A Transformer model estimates the score function (or vector field) based on $\bar{\mathbf{s}}(t)$, which is then used to update the state to $\bar{\mathbf{s}}(t-\Delta t)$. This iterative process continues until the clean state $\bar{\mathbf{s}}(0)$ is reached, which is finally decoded into discrete tokens. Note that the context embedding $\mathbf{e}_c$ remain fixed during updates.

although the diffusion and generation processes operate in the continuous space to obtain $\bar{\mathbf{s}}(0)$, the goal is to produce discrete tokens. As a result, an additional module is required to decode the generated continuous embeddings back into the discrete tokens. We briefly discuss these two challenges below.

5.2.2 Length Prediction

Length prediction is a recurring problem in NLP with a long history of research. A straightforward approach is to explicitly predict the target sequence length based on the context. For instance, the target length can be derived simply by multiplying the source length by a specific ratio. This concept dates back to the early era of **statistical machine translation** (SMT), where fertility models were employed to determine the number of target words corresponding to a single source word [Brown et al., 1993]. Similar mechanisms were adopted during the development of NAR generation models [Xiao et al., 2023]. In early NAR generation models (typically encoder-decoder architectures), a fertility predictor was integrated into the encoder. This predictor estimates the number of times each encoder hidden state should be copied to the decoder, thereby planning the target sequence length prior to decoding. In such methods, length prediction is framed as modeling the probability $\Pr(n|\mathbf{c})$. The predictor is typically a neural network, such as a **multi-layer perceptron** (MLP), trained to maximize the likelihood of the ground-truth length given the context. During inference, the target length is determined by selecting the length n^* with the highest probability.

Explicit length prediction is simple but often leads to repetition and hallucination, as the model attempts to fill the unused capacity of the pre-allocated window with meaningless or redundant tokens. Instead, most current diffusion language models adopt implicit length determination. This approach is consistent with the behavior of AR generation by allowing the model to signal its own termination. In practice, the generation process starts with a sufficiently long window (e.g., the maximum sequence length). Once this process is complete and the continuous embeddings have been decoded into discrete tokens, the system identifies the first occurrence of a termination marker (such as the EOS token). This position is treated as the logical end of the sentence, and any subsequent tokens are discarded.

Recent studies have also addressed the computational inefficiency of pre-allocating maximum-length windows by introducing dynamic length adaptation mechanisms [Li et al., 2025; Yang et al., 2025]. Rather than relying solely on post-hoc truncation via EOS tokens, these approaches integrate length modulation directly into the diffusion and generation processes. Through discrete insertion and deletion operations or flexible noise schedules, they can adjust the sequence length on-the-fly.

5.2.3 Rounding

As described above, the generation process terminates at $t = 0$, yielding a sequence of continuous vectors $\mathbf{s}(0)$. Let us denote the generated embedding sequence within $\bar{\mathbf{s}}(0)$ as $\hat{\mathbf{e}} = [\hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_n]$, where each $\hat{\mathbf{e}}_i \in \mathbb{R}^d$. Since our goal is to generate text, we must map these continuous vectors back to discrete tokens in the vocabulary V. This operation is commonly referred to as **rounding** or decoding.

The most straightforward rounding method is **nearest neighbor search**. Suppose we represent tokens as embeddings using a pre-trained embedding matrix $\mathbf{E} \in \mathbb{R}^{d \times |V|}$, where each column corresponds to the embedding of a token $v \in V$. Then, we can select the token whose embedding is closest to the generated vector $\hat{\mathbf{e}}_i$ under a specific distance metric. Since diffusion models are

typically trained using an MSE loss, the Euclidean distance is a natural choice

$$x_i \quad = \quad \arg\min_{v \in V} \|\hat{\mathbf{e}}_i - \mathbf{e}_v\|_2^2, \tag{11}$$

where $\mathbf{e}_v$ denotes the column of $\mathbf{E}$ corresponding to token v. Alternatively, one can employ a trainable projection layer followed by a softmax function, similar to the output layer in standard AR language models. In this case, the probability of token x_i is given by the Softmax function.

In contrast, for models based on latent or contextualized embeddings (e.g., those using VAEs or LLM representations), simple nearest neighbor search is inapplicable because the generated latent variables do not directly correspond to static token embeddings. In these approaches, a learnable decoder is required to map the continuous latent sequence back to the discrete token space. The decoding process usually involves a neural network that predicts the probability distribution over the vocabulary for each latent vector, as in AR generation models.

Rounding is not trivial due to the mismatch between the continuous diffusion latent space and the discrete nature of token embeddings. The standard diffusion training objective (such as MSE) encourages the model to predict the expected value of the data. In the context of language, if the distribution is multimodal (e.g., the next token could plausibly be "cat" or "dog"), the model might predict the average of their embeddings: $(\mathbf{e}_{\text{cat}} + \mathbf{e}_{\text{dog}})/2$. This averaged vector lies in the interstitial space between valid embeddings. Simply rounding this vector to the nearest neighbor may result in a semantically broad token (e.g., "animal"), or an entirely unrelated token if the embedding space is not perfectly smooth. This phenomenon is often observed as the generated vectors failing to commit to a specific discrete mode.

To mitigate this issue, researchers have introduced auxiliary objectives and architectural modifications. Li et al. [2022] proposed a trainable rounding parameter and added an auxiliary regularization term to the training objective. This term explicitly encourages the model to generate vectors that lie close to the valid embeddings in $\mathbf{E}$, thereby reducing the rounding error during inference. Strudel et al. [2022] introduced the concept of self-conditioning, where the estimate of the denoised token is projected back to the embedding space and used as input for the next step. This effectively clamps the intermediate states closer to the valid data manifold, and enables the final state $\mathbf{s}(0)$ to be reliably mapped to discrete tokens.

5.3 Diffusion in Token Space: Discrete Diffusion

While embedding-space diffusion fits well into the framework of standard diffusion models, it introduces the non-trivial challenge of rounding continuous vectors back to discrete tokens. To bypass this issue and align the generation process more naturally with the discrete nature of language, a growing body of research explores discrete diffusion. In this paradigm, the corruption process operates directly on the discrete vocabulary space V, thereby removing the need for embedding projection and rounding.

5.3.1 The CTMC Perspective

To model diffusion and generation processes directly on the discrete vocabulary V, we move away from the Gaussian noise and SDEs that are commonly used in continuous spaces. In a discrete state space, the process evolves in continuous time but changes state only via instantaneous jumps: it stays constant for a random holding time and then jumps to a new state. The mathematical framework that naturally describes such probabilistic jumps is the continuous-time Markov chain (CTMC). Just as the SDE framework views diffusion as the limit of a random walk as the step size approaches zero, the CTMC framework views discrete diffusion as the limit of a discrete-time Markov chain as the number of time steps approaches infinity.

Since the CTMC here operates on discrete tokens, we use $\mathbf{x}(t)$ instead of $\mathbf{s}(t)$ to denote a state representing a token sequence. Let $\{\mathbf{x}(t)\}_{t\geq 0}$ be a stochastic process where $\mathbf{x}(t)$ is a sequence of n tokens at time t. Let $\mathbf{x}$ and $\mathbf{y}$ denote specific states (i.e., token sequences) in the discrete space V^n. The evolution of the probability distribution $p_t(\mathbf{x}) = \Pr(\mathbf{x}(t) = \mathbf{x})$ is governed by the Kolmogorov forward equation

$$\frac{dp_t(\mathbf{x})}{dt} = \sum_{\mathbf{y}\in V^n} p_t(\mathbf{y})[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{yx}}, \tag{12}$$

where $\mathbf{Q}^{\text{seq}}(t)$ is the (time-dependent) generator matrix for the entire sequence. The term $[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{yx}}$ denotes the instantaneous rate of transitioning from state $\mathbf{y}$ to state $\mathbf{x}$, which is an entry of the large matrix $\mathbf{Q}^{\text{seq}}(t)$.

However, the state space size $|V|^n$ is exponentially large, making the full matrix $\mathbf{Q}^{\text{seq}}(t)$ computationally intractable. To address this, we generally assume independence between token positions during the forward process. Under this assumption, the transitions occur independently at each position. Thus, we can model the process using a much smaller single-token generator matrix $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V|\times|V|}$ that operates on individual tokens. As a result, the complex high-dimensional dynamics of $\mathbf{Q}^{\text{seq}}(t)$ can be factorized into n parallel processes, each governed by the same single-token generator $\mathbf{Q}^{\text{tok}}(t)$. Figure 4 shows an illustration of Eq. (12).

Note that, while CTMCs provide a continuous-time view via the forward equation in Eq. (12), practical discrete diffusion models in NLP typically operate on a discretized time grid $t \in \{0, 1, \ldots, T\}$. In this case, the process is implemented through transition kernels (or matrices) between adjacent time steps, and CTMC expressions can be understood as the continuous-time limit of these discretized dynamics.

Using CTMCs, we describe the forward and backward processes as follows:

- **The Forward Process**. In CTMCs, instead of adding Gaussian noise, we corrupt the text via stochastic token substitutions. This is defined by a predetermined generator matrix $\mathbf{Q}^{\text{tok}}(t)$. Commonly used corruption schemes include:
  - **Uniform Diffusion:** The token has a uniform rate of jumping to any other token in the vocabulary. This is analogous to the Gaussian noise in continuous space which drives the data distribution toward a standard normal distribution. Here, the distribution converges to a uniform distribution over V.

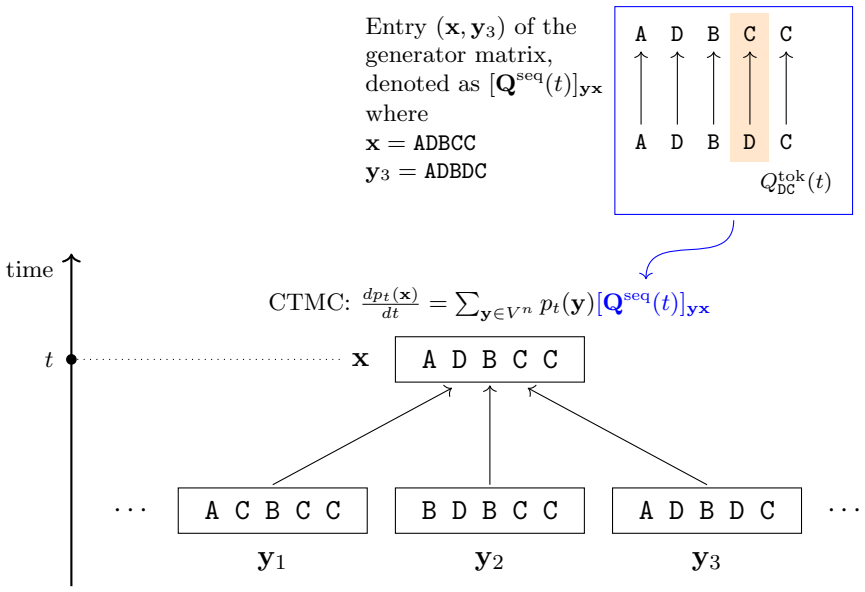


Figure 4: Illustration of the CTMC in token sequence generation. The dynamics of the CTMC at time t is $\frac{dp_t(\mathbf{x})}{dt} = \sum_{\mathbf{y}\in V^n} p_t(\mathbf{y})[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{yx}}$. It states that the rate of change of the probability of observing a target sequence $\mathbf{x}$ (e.g., ADBCC) is the sum of the probabilities from all possible source sequences (e.g., $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$), weighted by the sequence-level transition rate $[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{yx}}$. The global transition rate is determined by the underlying token-level dynamics. For example, a transition from sequence $\mathbf{y}_3$ to $\mathbf{x}$ involves individual token changes, such as the rate $Q^{\text{tok}}_{\text{DC}}(t)$ for the fourth token changing from D to C. Note that while the CTMC models dynamics in continuous time, the state transitions themselves are discrete jumps rather than continuous changes.

  - **Masking Diffusion (Absorbing State):** Tokens transition to a special [MASK] token with a certain rate but never transition back. This corresponds to an absorbing state in the Markov chain.

- **The Reverse Process**. A standard result in stochastic processes states that if the forward process is a CTMC with generator $\mathbf{Q}^{\text{tok}}(t)$, the reverse process running backward in time from T to 0 is also a CTMC [Anderson, 1982]. Its generative rate matrix $\tilde{\mathbf{Q}}^{\text{tok}}(t)$ is given by

$$\tilde{Q}^{\text{tok}}_{vu}(t) \quad = \quad Q^{\text{tok}}_{uv}(t)\frac{p_t(u)}{p_t(v)}, \tag{13}$$

  where $u, v \in V$ represent specific token values in the vocabulary. Here, $\tilde{Q}^{\text{tok}}_{vu}(t)$ denotes the rate of transitioning from value v to value u in the reverse process, and $p_t(v)$ is the marginal probability of a token taking the value v at time t.

The above equation establishes the fundamental connection between CTMCs and discrete diffusion models. It is the discrete analog of the reverse-time SDE formula, which relates the

reverse drift to the score function $\nabla \log p_t(\mathbf{x})$. In the discrete case, the probability ratio $p_t(u)/p_t(v)$ acts as the "discrete score", which guides the generation process toward high-probability tokens.

Eq. (13) is a theoretical characterization of the time-reversed CTMC, where the reverse rates depend on the true marginals $p_t(\cdot)$. In discrete diffusion models for text, we typically do not estimate $p_t(\cdot)$ explicitly. Instead, the reverse dynamics are constructed through a Bayes factorization of the reverse posterior $\Pr(\mathbf{x}(t-\Delta t) \mid \mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$. Thus, Eq. (13) mainly serves to provide intuition for why probability ratios or posterior reweighting guide the reverse process toward high-probability tokens. In practice, we parameterize the reverse dynamics with a neural network (typically a Transformer) that predicts a denoising distribution such as $p_\theta(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c})$ [3].

A representative example in CTMC-based diffusion modeling is the **structured denoising diffusion probabilistic model** (D3PM) proposed by Austin et al. [2021]. While earlier attempts at discrete diffusion often relied on simple uniform transition probabilities, D3PM generalizes the framework to support arbitrary noise processes through structured transition matrices. Importantly, Austin et al. [2021] demonstrated that the choice of the corruption process significantly impacts sample quality. They introduced several structured transition schemes beyond standard uniform diffusion. In terms of model parameterization, D3PM is different from the noise-prediction paradigm. Since noise is not additive in discrete space, D3PM is trained to predict the clean data distribution $p_\theta(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c})$ directly from the noisy input.

Note that since the ground truth $\mathbf{x}(0)$ is available during training, we can directly compute the true reverse posterior $q(\mathbf{x}(t-\Delta t)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$. Consequently, this true posterior serves as the supervision signal for the model's transition probabilities.

Formally, this reverse posterior corresponds to the distribution over the previous time step. Following common notation in diffusion models, we denote the forward conditional distribution as q. Then, the reverse posterior can be derived using Bayes' rule:

$$q(\mathbf{x}(t-\Delta t)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c}) \quad = \quad \frac{q(\mathbf{x}(t)|\mathbf{x}(t-\Delta t), \mathbf{c})\, q(\mathbf{x}(t-\Delta t)|\mathbf{x}(0), \mathbf{c})}{q(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{c})}. \tag{14}$$

Here, the probability distributions over the sequence factorize into independent per-token transitions, e.g.,

$$q(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{c}) \quad = \quad \prod_{i=1}^{n} q(x_i(t)|x_i(0), \mathbf{c}), \tag{15}$$

where each probability $q(x_i(t)|x_i(0), \mathbf{c})$ is determined by the single-token generator $\mathbf{Q}^{\text{tok}}(t)$[4]. In practice, for structured $\mathbf{Q}^{\text{tok}}(t)$ matrices like uniform and masking diffusion, the forward transition

3. In most settings, the forward corruption process is chosen independent of $\mathbf{c}$. We keep $\mathbf{c}$ in the notation to match the conditional generation setting and to allow the more general case where the noise process may depend on $\mathbf{c}$.
4. To be precise, $q(x_i(t)|x_i(0), \mathbf{c})$ corresponds to the entry $(x_i(0), x_i(t))$ of the single-token transition probability matrix $\mathbf{P}^{\text{tok}}(0,t) = \exp\left(\int_0^t \mathbf{Q}^{\text{tok}}(\tau)\, d\tau\right)$ (assuming commutativity over time). Physically, $\mathbf{P}^{\text{tok}}(0,t)$ represents the total probability mass transferred between tokens over the interval $[0,t]$. Due to the Markov property, this matrix can also be viewed as the product of transition matrices over consecutive discrete time steps: if we discretize the interval into $0 = t_0 < t_1 < \cdots < t_k = t$, then $\mathbf{P}^{\text{tok}}(0,t) = \mathbf{P}^{\text{tok}}(t_{k-1}, t_k) \times \cdots \times \mathbf{P}^{\text{tok}}(t_0, t_1)$.

probability $q(x(t)|x(0), \mathbf{c})$ has a simple analytical closed-form solution. For uniform diffusion, we have

$$q(x(t)|x(0), \mathbf{c}) \quad = \quad \alpha_t \cdot \mathbb{I}(x(t) = x(0)) + (1 - \alpha_t) \cdot \frac{1}{|V|}, \tag{16}$$

where α_t is a time-dependent scalar schedule parameter that controls the signal retention rate. Similarly, for the absorbing state (masking) diffusion, the probability mass is directed towards a special `[MASK]` token. The corresponding closed-form transition probability is given by

$$q(x(t)|x(0), \mathbf{c}) \quad = \quad \begin{cases} \alpha_t, & \text{if } x(t) = x(0), \\ 1 - \alpha_t, & \text{if } x(t) = \texttt{[MASK]}, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

These analytical solutions allow us to bypass the computationally expensive matrix operations in the forward process. With these tractable forward probabilities, the backward process is realized by substituting the neural network's prediction of the original token sequence, $p_\theta(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c})$, into the Bayesian factorization derived above. This yields a reverse transition distribution $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ from which we can sample. Figure 5 illustrates the forward and backward processes in the D3PM with uniform diffusion, assuming $\Delta t = 1$.

To train the model, we wish to maximize the log-likelihood of the observed data $\log p_\theta(\mathbf{x}(0)|\mathbf{c})$. However, this requires marginalizing over all possible trajectories $\mathbf{x}(1{:}T)$, which is intractable. By introducing the forward process $q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c})$ as an auxiliary distribution, we can rewrite the log marginal probability as an expectation

$$\begin{aligned} \log p_\theta(\mathbf{x}(0)|\mathbf{c}) \quad &= \quad \log \sum_{\mathbf{x}(1:T)} q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c}) \frac{p_\theta(\mathbf{x}(0{:}T)|\mathbf{c})}{q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c})} \\ &= \quad \log \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0),\mathbf{c})} \left[\frac{p_\theta(\mathbf{x}(0{:}T)|\mathbf{c})}{q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c})} \right]. \end{aligned} \tag{18}$$

Applying Jensen's inequality, we move the logarithm inside the expectation to derive a tractable **evidence lower bound** (ELBO):

$$\log p_\theta(\mathbf{x}(0)|\mathbf{c}) \quad \geq \quad \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0),\mathbf{c})} \left[\log \frac{p_\theta(\mathbf{x}(0{:}T)|\mathbf{c})}{q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c})} \right]. \tag{19}$$

By expanding both the joint distribution $p_\theta(\mathbf{x}(0{:}T)|\mathbf{c})$ and the forward trajectory $q(\mathbf{x}(1{:}T)|\mathbf{x}(0), \mathbf{c})$ into products of their respective conditional distributions, and then rearranging terms, the negative ELBO can be decomposed into a sum of per-step KL divergence terms (see [Austin et al., 2021] for the full derivation in the discrete setting). The resulting training objective, which the

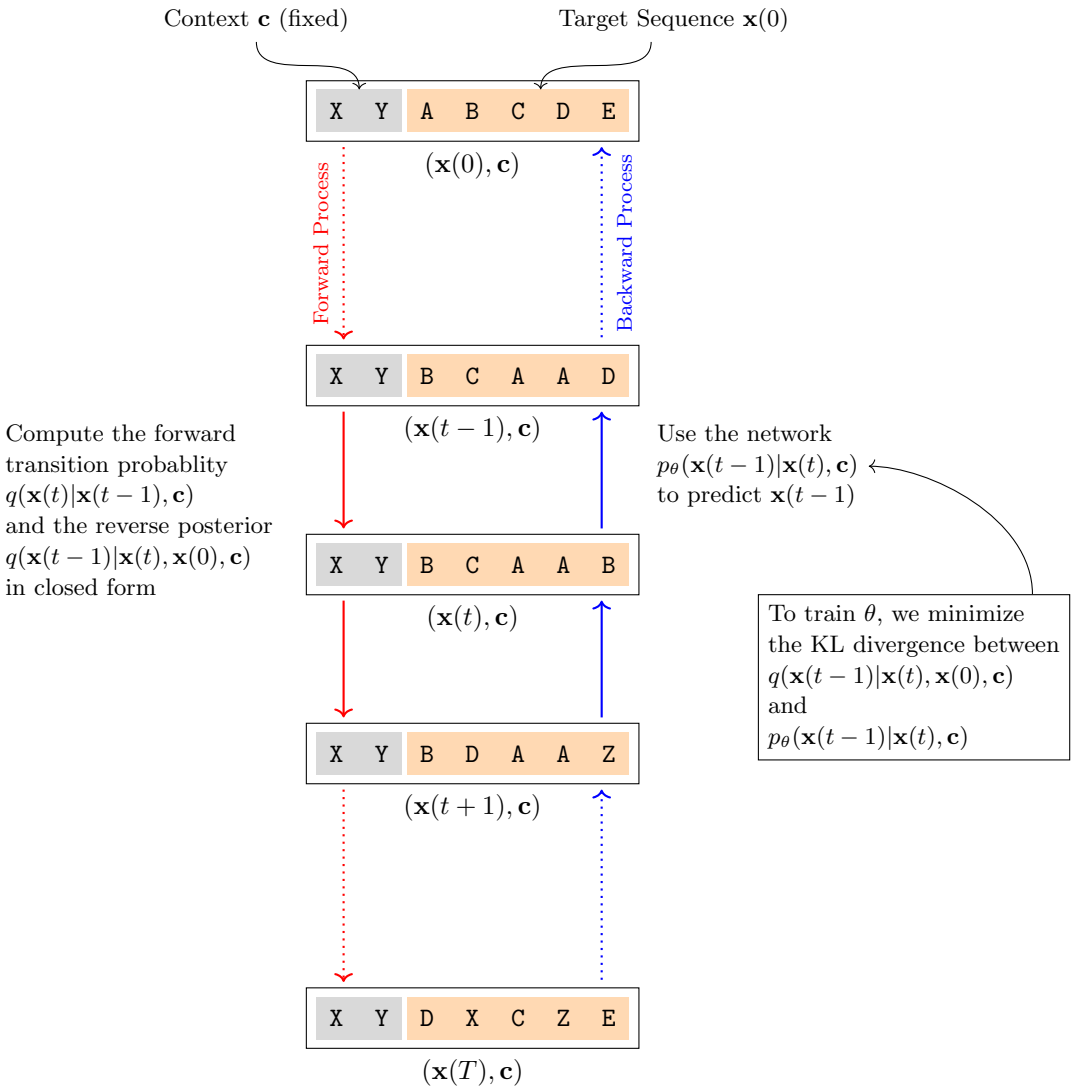


Figure 5: Illustration of the forward and backward processes in the D3PM with uniform diffusion. During the forward process, a state jumps to a new state at each time step of $\Delta t = 1$, with the initial state $(\mathbf{x}(0), \mathbf{c})$ and the final state $(\mathbf{x}(T), \mathbf{c})$. In a forward step, we compute the forward transition probability $q(\mathbf{x}(t)|\mathbf{x}(t-1), \mathbf{c})$, which can then be used to compute the reverse posterior $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ for training the backward model. During the backward process, the model evolves in reverse time. At each backward step, we use the network $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ to predict the token sequence $\mathbf{x}(t-1)$, and then move on to $t-1$. The network is trained to minimize the KL divergence between $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ and $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ in expectation over q.

model minimizes, is given by

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0),\mathbf{c})}\Bigg[-\log p(\mathbf{x}(T)) + \sum_{t=1}^{T} \text{KL}\Big(q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c}) \,\|\, p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})\Big)\Bigg], \quad (20)$$

In this objective, the term $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ represents the true posterior of the reverse transition. A key advantage here is that, for structured forward processes (like uniform noise or masking), this posterior distribution can be efficiently computed in closed form using Bayes' rule. Regarding the first term, since the prior $p(\mathbf{x}(T))$ is typically a fixed distribution (e.g., a uniform distribution or a deterministic masked state) with no learnable parameters, $-\log p(\mathbf{x}(T))$ is constant with respect to θ and can be omitted. As a result, the core learning signal drives the model p_θ to align its reverse transition probabilities with the ideal reverse path dictated by the forward dynamics.

5.3.2 Masked Diffusion Models

Masked diffusion models (MDMs) represent a specialized implementation of discrete diffusion where the transition rate matrix $\mathbf{Q}^{\text{tok}}(t)$ is designed to move probability mass toward a special absorbing state, denoted as the `[MASK]` token [Sahoo et al., 2024; Nie et al., 2025]. This approach provides a unique practical advantage: once a token transitions to the masked state in the forward process, it remains there, and thus creates a unidirectional path toward a fully corrupted state.

The forward process progressively replaces tokens in the original sequence with `[MASK]`. It starts from a fully observed sequence and gradually destroys information until the entire sequence becomes a mask-only sequence. In practice, we discretize time and implement the process as a discrete-time Markov chain on discrete time steps $\{0, 1, \ldots, T\}$. Let $\mathbf{x}(t)$ be the corrupted sequence at time $t \in \{0, 1, \ldots, T\}$ with $\mathbf{x}(0)$ being clean data and $\mathbf{x}(T) = \texttt{[MASK]}^n$, and $x_i(t)$ be the i-th token of $\mathbf{x}(t)$. Under the standard independence assumption across positions in the forward process, masking diffusion can be written as a per-token absorbing transition

$$q(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{c}) \quad = \quad \prod_{i=1}^{n} q(x_i(t)|x_i(0), \mathbf{c}), \tag{21}$$

where the single-token forward kernel is

$$q(x_i(t)|x_i(0), \mathbf{c}) \quad = \quad \alpha_t \cdot \mathbb{I}(x_i(t) = x_i(0)) + (1 - \alpha_t) \cdot \mathbb{I}(x_i(t) = \texttt{[MASK]}). \tag{22}$$

Here $\alpha_t \in [0, 1]$ is a monotonically decreasing schedule that controls how much signal is retained at time t. Equivalently, we may view $\mathbf{x}(t)$ as produced by sampling a binary mask indicator vector $\mathbf{m}_t \in \{0, 1\}^n$ with

$$m_{t,i} \sim \text{Bernoulli}(1 - \alpha_t), \tag{23}$$

$$x_i(t) = \begin{cases} \texttt{[MASK]}, & m_{t,i} = 1, \\ x_i(0), & m_{t,i} = 0. \end{cases} \tag{24}$$

Thus the diffusion trajectory can be interpreted as repeatedly increasing the set of masked positions until eventually all positions are masked, as illustrated in Figure 6.

The backward process runs from $t = T$ to $t = 0$. Given the context $\mathbf{c}$ and an initial fully masked sequence $\mathbf{x}(T) = \texttt{[MASK]}^n$, the MDM iteratively replaces `[MASK]` tokens with real tokens

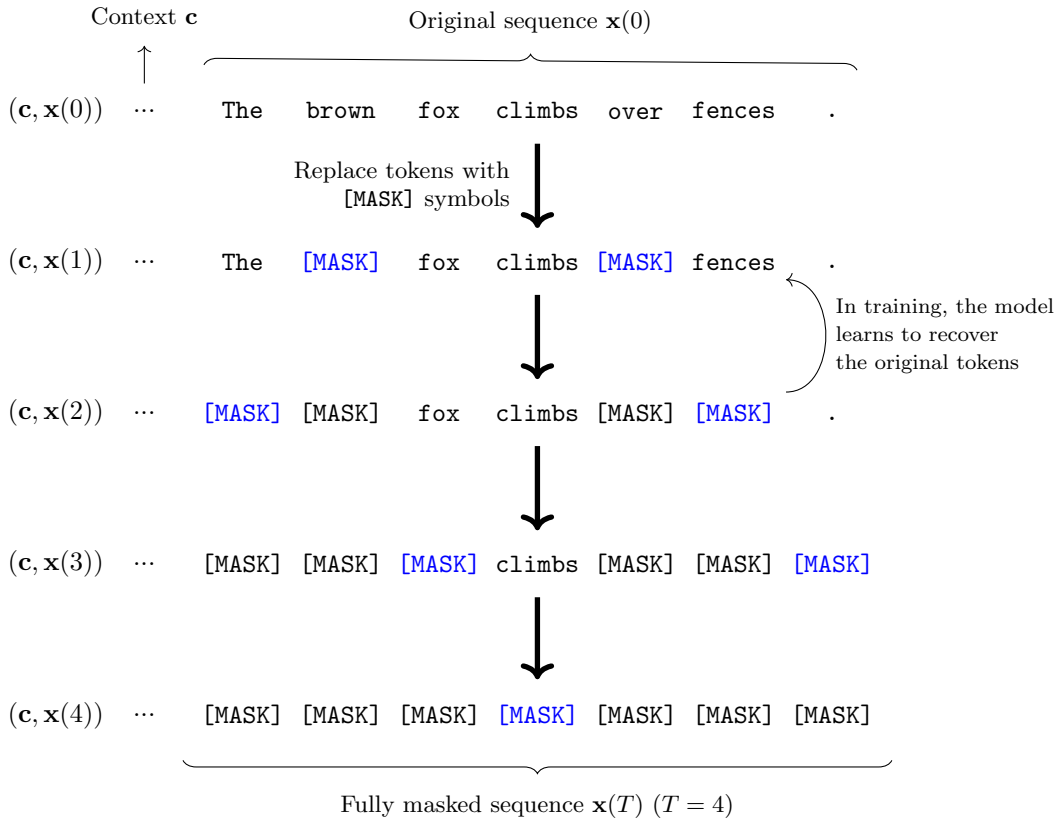


Figure 6: Illustration of the forward process in MDMs, where an original token sequence is iteratively transformed into a fully masked sequence. During this process, tokens are replaced with `[MASK]` symbols until the entire sequence is masked. Note that the context $\mathbf{c}$ remains unmasked throughout. To train an MDM, the model learns to recover the masked tokens by conditioning on both the fixed context $\mathbf{c}$ and the partially corrupted sequence.

until obtaining a fully instantiated sequence in V^n. Formally, we define a parameterized reverse Markov chain

$$p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c}), \quad t = T, T-1, \ldots, 1, \tag{25}$$

Here $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ is the notation of the reverse conditional distribution $\Pr_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$, where the subscript θ emphasizes that the reverse-time dynamics are parameterized by θ.

A simple parameterization is to let the network predict the denoising distribution over the clean tokens,

$$p_\theta(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c}) = \prod_{i=1}^{n} p_\theta(x_i(0)|\mathbf{x}(t), \mathbf{c}). \tag{26}$$

Then, we construct the reverse transition by only updating masked positions, while keeping unmasked tokens unchanged. For each position i, we have

$$p_\theta(x_i(t-1)|\mathbf{x}(t),\mathbf{c}) = \begin{cases} \mathbb{I}(x_i(t-1)=x_i(t)), & \text{if } x_i(t) \neq \texttt{[MASK]}, \\ \pi_{\theta,t,i}(\cdot|\mathbf{x}(t),\mathbf{c}), & \text{if } x_i(t) = \texttt{[MASK]}, \end{cases} \tag{27}$$

where $\pi_{\theta,t,i}(\cdot|\mathbf{x}(t),\mathbf{c})$ is a categorical distribution on V produced by the model. A common choice is

$$\pi_{\theta,t,i}(v|\mathbf{x}(t),\mathbf{c}) = p_\theta(x_i(0)=v|\mathbf{x}(t),\mathbf{c}), \quad v \in V, \tag{28}$$

i.e., directly sample the missing token from the predicted clean-token distribution.

Given a dataset of examples $(\mathbf{x}(0),\mathbf{c}) \sim p_{\text{data}}$, MDMs learn the reverse model p_θ by maximizing the conditional likelihood of the clean sequence, or equivalently minimizing its negative log-likelihood. Similar to other diffusion models, we can train MDMs using the ELBO (see also D3PM in Section 5.3.1).

Although the above training objective appears a bit tedious, in practice, a simpler and widely used approach is to train the model to recover the original tokens at masked positions. We sample a time step $t \sim U\{1,\dots,T\}$ and then sample a corrupted sequence $\mathbf{x}(t) \sim q(\mathbf{x}(t)|\mathbf{x}(0),\mathbf{c})$. We define the masked index set to be

$$\mathcal{M}(t) \quad = \quad \{i \in \{1,\dots,n\} : x_i(t) = \texttt{[MASK]}\}. \tag{29}$$

The loss is then defined as the conditional negative log-likelihood on masked positions, as follows

$$\begin{aligned} & \mathcal{L}_{\text{denoise}}(\theta) \\ = \quad & \mathbb{E}_{(\mathbf{x}(0),\mathbf{c})\sim p_{\text{data}}}\, \mathbb{E}_{t\sim U[T]}\, \mathbb{E}_{\mathbf{x}(t)\sim q(\mathbf{x}(t)|\mathbf{x}(0),\mathbf{c})} \Bigg[-\sum_{i\in\mathcal{M}(t)} \log p_\theta(x_i(0)|\mathbf{x}(t),\mathbf{c}) \Bigg]. \end{aligned} \tag{30}$$

Optionally, one can reweight time steps to emphasize certain noise levels:

$$\mathcal{L}_{\text{denoise}}(\theta) \quad = \quad \mathbb{E}\Bigg[\gamma_t \cdot \Big(-\sum_{i\in\mathcal{M}(t)} \log p_\theta(x_i(0)|\mathbf{x}(t),\mathbf{c})\Big)\Bigg], \tag{31}$$

where $\gamma_t \geq 0$ is a predefined schedule. Intuitively, as in curriculum learning, small t corresponds to light corruption (i.e., easier denoising), while large t corresponds to heavy corruption (i.e., harder denoising). The weighting γ_t can be tuned to balance these regimes.

After training, we use the learned reverse kernel $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t),\mathbf{c})$ to generate text by running the Markov chain backward from a fully masked state. The inference starts from

$$\mathbf{x}(T) = \texttt{[MASK]}^n, \tag{32}$$

and iteratively samples

$$\mathbf{x}(t-1) \sim p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c}), \quad t = T, T-1, \ldots, 1. \tag{33}$$

Under the per-token factorization in Eq. (27), the reverse step only updates masked positions. For each i, we have

$$x_i(t-1) = \begin{cases} x_i(t), & x_i(t) \neq \texttt{[MASK]}, \\ \mathrm{Cat}(\pi_{\theta,t,i}(\cdot|\mathbf{x}(t), \mathbf{c})), & x_i(t) = \texttt{[MASK]}, \end{cases} \tag{34}$$

where $\mathrm{Cat}(\cdot)$ denotes sampling a discrete token from the categorical distribution specified by its argument, i.e., $x_i(t-1) \sim \pi_{\theta,t,i}(\cdot|\mathbf{x}(t), \mathbf{c})$, when $x_i(t) = \texttt{[MASK]}$. Thus, the model repeatedly fills in missing tokens conditioned on the already-filled context, until all positions become unmasked.

The above approach does not specify how many tokens should be unmasked per step. In practice, MDMs typically adopt an explicit unmasking schedule. Let k_t be the number of tokens to be newly unmasked when going from t to $t-1$, and let $\mathcal{U}(t) \subseteq \mathcal{M}(t)$ be the selected subset of masked positions to update, with $|\mathcal{U}(t)| = k_t$. Then a common implementation is

$$x_i(t-1) = \begin{cases} x_i(t), & i \notin \mathcal{U}(t), \\ \mathrm{Cat}(\pi_{\theta,t,i}(\cdot|\mathbf{x}(t), \mathbf{c})), & i \in \mathcal{U}(t). \end{cases} \tag{35}$$

The schedule $\{k_t\}_{t=1}^T$ trades off speed and accuracy: larger k_t leads to fewer iterations but requires the model to fill many tokens under high uncertainty, whereas smaller k_t yields more refinement steps but increases compute.

A simple choice is a deterministic schedule that reveals a fixed fraction each step (e.g., linear or cosine in t), such that the expected number of masked tokens decreases from n to 0. Another common strategy is to pick $\mathcal{U}(t)$ based on model confidence. Given $\pi_{\theta,t,i}$, we can define a confidence score as

$$s_{t,i} = \max_{v \in V} \ \pi_{\theta,t,i}(v|\mathbf{x}(t), \mathbf{c}), \tag{36}$$

Then we can unmask positions with the highest confidence first. This easy-first heuristic tends to stabilize generation by anchoring the sequence with reliable tokens before filling more ambiguous ones.

5.4 Remarks on Discrete Diffusion

In this subsection, we discuss several modeling considerations for discrete diffusion in token space.

5.4.1 Rethinking Non-absorbing Replacement Diffusion

The absorbing `[MASK]` construction has an appealing information monotonicity: the forward process only removes information, and the reverse process only fills it back in. This design aligns well with Transformer denoising and has recently demonstrated strong scalability in large-scale

pretraining of diffusion language models [Sahoo et al., 2024; Nie et al., 2025]. Nevertheless, it is valuable to revisit non-absorbing replacement diffusion methods (such as uniform diffusion), where tokens may be repeatedly substituted throughout the trajectory.

First, uniform replacement is a simple and well-behaved baseline. The uniform substitution kernel used in D3PM is the canonical example. At each step, a token is either kept or replaced by a uniformly sampled vocabulary element. This strategy yields closed-form transition probabilities, thereby keeping training tractable. Moreover, because the corruption is reversible in principle, the reverse process is not constrained by an irreversible absorbing sink. Conceptually, uniform replacement is the discrete analog of adding isotropic Gaussian noise. It drives the distribution toward a simple prior while preserving analytical control of the forward marginals.

Second, the reverse updates are dense in non-absorbing replacement diffusion. In masking diffusion, unmasked tokens are typically preserved and only masked positions are updated. In non-absorbing replacement diffusion, every position remains stochastic, so a natural reverse kernel can update all tokens at each step. This resembles the structure of continuous-time diffusion models, where all dimensions are refined throughout the trajectory. Rather than treating visible coordinates as fixed, the dynamics can revise any component to improve global consistency.

Third, non-absorbing replacement can be difficult in language. If tokens can be corrupted multiple times, the noisy sequence $\mathbf{x}(t)$ may contain locally fluent but globally inconsistent fragments, and the reverse model must both infer missing semantics and correct erroneous tokens. This is in contrast to masking diffusion, where generated tokens are typically clean. From the CTMC viewpoint, non-absorbing replacement corresponds to a generator $\mathbf{Q}^{\mathrm{tok}}(t)$ with nonzero off-diagonal rates between all token pairs, so that probability mass continuously circulates on V instead of flowing into a single sink. As t increases, the signal-to-noise ratio can degrade faster than in absorbing diffusion, because even observed tokens are unreliable.

Recent work has explored methods of combining absorbing masking with additional replacement noise to induce a corrective behavior. In this way, the model can detect and fix corrupted tokens rather than only fill masked positions [Zhang et al., 2025; Rütte et al., 2025]. This hybrid view suggests that non-absorbing noise is not merely a competitor to masking diffusion, but can be a complementary mechanism for robustness and self-correction.

A simple idea is to consider structured replacement strategies. In NLP, a natural method is to bias replacement using token-dependent or structure-aware transition rates, for example,

$$Q_{uv}^{\mathrm{tok}}(t) \quad = \quad w_{uv}(t), \tag{37}$$

where $w_{uv}(t)$ models confusability which could be frequency-aware, subword-related, or embedding-similarity-based. The motivation is to make early noise semantically local (easy to denoise) and late noise globally mixing (approaching a simple prior). This is similar to variance-preserving vs. variance-exploding design choices in continuous diffusion, but now expressed as a schedule over discrete transition structure.

Another interesting direction is to interpolate between masking-style absorbing diffusion and uniform replacement diffusion. For example, one may consider a combination at the generator

level:

$$\mathbf{Q}^{\text{tok}}(t) = \lambda_t \cdot \mathbf{Q}^{\text{tok}}_{\text{mask}}(t) + (1 - \lambda_t) \cdot \mathbf{Q}^{\text{tok}}_{\text{unif}}(t), \tag{38}$$

where the model behaves like masking diffusion at certain noise levels while retaining the ability to distribute probability mass among unmasked tokens at others. Such a combination has recently been explored as a way to obtain the best of both regimes (stability from absorbing corruption and flexibility from replacement corruption).

5.4.2 Masked Diffusion Modeling vs Masked Language Modeling

In the general framework of corruption-and-denoise, the training objective of masked diffusion models is very close to that of **masked language modeling** (MLM). For example, in BERT-like models, one samples a mask pattern and trains a bidirectional Transformer to predict the original tokens at masked positions from partially observed context [Devlin et al., 2019]. If we interpret Eq. (24) as sampling a mask indicator $\mathbf{m}_t$ with mask rate $(1 - \alpha_t)$, then the denoising loss in Eq. (30) is essentially an MLM cross-entropy, except that the corruption level is explicitly indexed by time through α_t and the network is conditioned on the diffusion step t. The main conceptual difference is therefore not the supervised signal but the deployment of the denoiser. BERT's MLM is primarily a pretraining objective for representation learning, whereas masked diffusion models interpret the same denoising primitive as a reverse-time transition kernel and apply it iteratively from a fully masked state to generate a natural token sequence. In this sense, modern masked diffusion models often look like BERT in their architectures, but they turn MLM from a pretraining task into a procedure for sequence generation.

The connection between masked diffusion models and MLM suggests an interesting direction: since the forward process in masked diffusion is a particular choice of corruption, many noising strategies developed for NLP pretraining can be reinterpreted as alternative forward kernels for discrete diffusion. That is, beyond independent token masking, one can vary what gets corrupted (tokens vs. spans), how it is corrupted (masking vs. replacement vs. permutation), and how the corruption level evolves with time (hand-designed vs. learned schedules). Below are some common noising strategies that can be considered in designing masked diffusion models.

- **Token replacement with non-uniform distributions.** Instead of replacing tokens uniformly, one can sample replacements from frequency-adjusted distributions or from a learned proposal. An example is ELECTRA, where a small generator proposes plausible replacements and a discriminator is trained to detect them [Clark et al., 2019]. From the diffusion perspective, such harder non-uniform replacements can be viewed as injecting more semantically confusable noise than random tokens, which may encourage stronger conditional denoisers and reduce the gap between training corruptions and inference-time uncertainty.

- **Span corruption.** The T5 series models replace contiguous spans with a single sentinel token and learn to reconstruct the missing content [Raffel et al., 2020]. For diffusion-style generation, span corruption is natural: the reverse process can be interpreted as progressively refining coarse missing regions into detailed text. This matches the intuition of iterative denoising over structured slots rather than independent token blanks. However, a

challenge with this approach is that replacing multiple tokens with a single sentinel alters the sequence length, which violates the fixed-dimension assumption of standard diffusion models. To address this, we can adopt in-place span masking similar to SpanBERT [Joshi et al., 2020], where contiguous tokens are masked but the sequence length remains constant. Alternatively, one can extend the forward kernel to explicitly model insertion and deletion operations to handle variable-length trajectories [Gu et al., 2019; Reid et al., 2022].

- **Token shuffling and permutation noise.** BART introduces denoising pretraining with corruptions such as token deletion and sentence permutation. This method requires the model to learn not only content restoration but also reordering [Lewis et al., 2020]. Recasting this as a diffusion forward kernel suggests a broader class of discrete dynamics that perturb both identity and position information. Such a method can lead to reverse processes that perform joint "edit + reorder" refinement rather than pure infilling.

- **Adversarial noising.** Another direction is to generate more challenging corruptions using model-guided perturbations or auxiliary networks. Adversarial training methods for NLP (e.g., embedding-level perturbations) show that stronger, structured noise can improve robustness [Zhu et al., 2020]. While many adversarial approaches are defined in continuous embedding space, the core idea is applicable to diffusion modeling. We can choose corruptions that maximally stress the denoiser at each noise level, which in principle can make the learned reverse dynamics more stable under iterative sampling.

- **Learned noising schedules.** Rather than fixing the schedule α_t by hand, one can attempt to learn how much and what type of noise to apply at each step, for example, by optimizing a combination of objectives or selecting corruption regimes that best serve downstream use [Tay et al., 2023]. In diffusion modeling, this motivates learning time-inhomogeneous corruption policies that shape the trajectory for better sample quality or fewer generation steps.

5.4.3 Simplex and Logit Dynamics as Continuous Relaxations

We have seen that the CTMC formulation frames discrete diffusion as distributional dynamics. Although the sample path $\mathbf{x}(t) \in V^n$ jumps among categorical states, the marginal $p_t(\cdot)$ evolves smoothly over t through the Kolmogorov equation in Eq. (12). This observation motivates a useful relaxation for language. Instead of treating the intermediate state as a hard token sequence, we represent each position by a probability column vector on the simplex,

$$\mathbf{p}_i(t) \in \Delta^{|V|-1}, \tag{39}$$

$$\mathbf{p}_i(t)[v] = \Pr(x_i(t) = v \mid \mathbf{c}). \tag{40}$$

We then aggregate these vectors into a matrix $\mathbf{P}(t) = [\mathbf{p}_1(t), \ldots, \mathbf{p}_n(t)] \in \mathbb{R}^{|V| \times n}$, where each column lies on the simplex. This converts the discrete trajectory into a continuous curve on a product of simplices. Then, we can apply ODE/SDE tools while still respecting the categorical nature of tokens.

For a token-level CTMC with generator $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V|\times|V|}$, the marginal distribution of a single position satisfies the Kolmogorov Forward equation

$$\frac{d\mathbf{p}_i(t)}{dt} = \left[\mathbf{Q}^{\text{tok}}(t)\right]^\top \mathbf{p}_i(t), \tag{41}$$

which is an ODE on the simplex. Note that the rows of $\mathbf{Q}^{\text{tok}}(t)$ sum to zero, so the total probability is conserved. Under the standard independence assumption across positions in the noising process, each $\mathbf{p}_i(t)$ evolves according to the same ODE. This gives an ODE viewpoint in language: discrete diffusion can be seen as choosing a linear forward vector field on the simplex, induced by $\mathbf{Q}^{\text{tok}}(t)$, rather than a Gaussian SDE in $\mathbb{R}^d$.

Working directly with probabilities can be numerically inconvenient near the boundary of the simplex (e.g., one-hot corners). A common alternative is to parameterize $\mathbf{p}_i(t)$ by logits $\mathbf{z}_i(t) \in \mathbb{R}^{|V|}$

$$\mathbf{p}_i(t) = \text{Softmax}(\mathbf{z}_i(t)). \tag{42}$$

We can then define a continuous-time dynamics in logit space,

$$\frac{d\mathbf{z}_i(t)}{dt} = \mathbf{v}_\theta\big(\mathbf{z}_i(t), t, \mathbf{c}\big), \tag{43}$$

where $\mathbf{v}_\theta$ is a Transformer-defined vector field (shared across positions). Intuitively, $\mathbf{z}_i(t)$ behaves like a continuously refined "soft token", and the terminal decoding $\mathbf{z}_i(0) \mapsto x_i$ can be implemented by arg max from $\text{Softmax}(\mathbf{z}_i(0))$. Relaxations of categorical sampling such as the Gumbel-Softmax distribution can also be used when differentiable sampling is desired [Jang et al., 2017; Maddison et al., 2017].

This logit view clarifies why continuous relaxations often feel similar to embedding-space diffusion, yet remain closer to discrete modeling: the state stays vocabulary-aligned (dimension $|V|$ per position) rather than living in an arbitrary embedding space $\mathbb{R}^d$. It also offers a direct connection to the CTMC marginal ODE in Eq. (41). A chosen $\mathbf{Q}^{\text{tok}}(t)$ implies a specific evolution of $\mathbf{p}_i(t)$, which can be re-expressed as an evolution in $\mathbf{z}_i(t)$ via the Softmax map.

As presented in Section 4, in continuous diffusion, the reverse-time dynamics can be expressed either as an SDE involving the score $\nabla \log p_t(\mathbf{s})$ or as a probability-flow ODE. In discrete diffusion, the exact reverse CTMC rates depend on probability ratios $p_t(u)/p_t(v)$ (Eq. (13)), which play the role of a discrete score. When we lift the state to $\mathbf{p}(t)$ or $\mathbf{z}(t)$, we can similarly interpret the reverse process as learning a vector field that flows from a simple prior (e.g., all-masked or uniform) to a distribution concentrated near one-hot corners representing real text. Recent discrete score-based formulations show that one can define tractable training objectives that avoid explicitly computing $p_t(\cdot)$ while still producing principled reverse-time dynamics [Hoogeboom et al., 2021; Lou et al., 2024].

To summarize, we outline the practical benefits of the simplex and logit relaxations, as follows

- **More efficient ODE solvers and fewer steps.** Once the reverse dynamics are expressed as an ODE like Eq. (43), we can use adaptive ODE solvers or coarse discretizations to

reduce the number of sampling steps, echoing the acceleration strategies used in continuous diffusion and flow models (Section 4.4).

- **Compatibility with flow matching.** On the simplex, flow matching naturally corresponds to learning $\mathbf{v}_\theta$ that transports soft token distributions toward sharp, data-like categorical distributions, without requiring an explicit likelihood-based ELBO at training time.

- **A spectrum between discrete and continuous.** Masked diffusion (hard `[MASK]` states) and embedding diffusion (continuous embeddings) can be seen as two endpoints. Simplex/logit dynamics occupy a middle ground. The state is continuous but remains directly interpretable as token probabilities, and the final projection to text is well-defined.

5.4.4 Relation to Iterative Refinement

The reverse process of discrete diffusion is closely related to a line of research on **iterative refinement** for non-autoregressive generation [Lee et al., 2018]. In iterative refinement, the model does not obtain a final sequence in one pass. Instead, it repeatedly generates a full sequence in parallel and then revises parts of it over multiple refinement rounds. An example is the mask-predict method for conditional masked language models, which starts from an all-`[MASK]` template and alternates between parallel prediction of tokens and re-masking a subset of uncertain positions, until convergence or a fixed number of iterations [Ghazvininejad et al., 2019].

This method can be expressed in a form that is similar to Eq. (35). Let $\mathbf{x}^{(r)}$ denote the sequence after r refinement rounds and let $\pi^{(r)}_{\theta,i}(v|\mathbf{x}^{(r)}, \mathbf{c}) = p_\theta(x_i^{(r+1)} = v|\mathbf{x}^{(r)}, \mathbf{c})$ be the predicted distribution of round r at position i. A refinement step can be expressed as

$$x_i^{(r+1)} \quad = \quad \begin{cases} x_i^{(r)}, & i \notin \mathcal{U}^{(r)}, \\ \mathrm{Cat}\left(\pi^{(r)}_{\theta,i}(\cdot|\mathbf{x}^{(r)}, \mathbf{c})\right), & i \in \mathcal{U}^{(r)}, \end{cases} \tag{44}$$

where $\mathcal{U}^{(r)}$ is the set of positions to update at round r. Typically, $\mathcal{U}^{(r)}$ is defined by low confidence, while in simple diffusion implementations it is often defined by a time schedule. In other words, masked diffusion sampling can be viewed as a structured refinement procedure in which the masking pattern evolves monotonically with time t (i.e., iteration index in iterative refinement), whereas classic refinement methods allow non-monotone decisions (e.g., re-masking already filled tokens) based on the uncertainty of the model. This difference matters: monotone unmasking is stable but may freeze early mistakes. By contrast, non-monotone refinement can correct errors but risks oscillation if the selection rule is not well defined.

Recent diffusion language models have begun to explicitly incorporate the non-monotone correction behavior that is standard in iterative refinement. One approach is to re-mask low-confidence tokens during inference, and combine diffusion sampling with mask-predict-like revisiting [Koh et al., 2025; Zhang et al., 2025]. Another approach is to modify the forward corruption so that the reverse model must learn not only to fill blanks but also to fix visible but wrong tokens. This can be achieved by mixing absorbing masking noise with replacement noise (e.g., uniform replacement), which yields a reverse process that naturally supports correction and editing [Rütte

et al., 2025]. These approaches overlap those presented in Section 5.4.1, and echo our discussion of combining different diffusion strategies.

We can also see iterative refinement from the perspective of diffusion modeling. The simplex and logit relaxations in Eqs. (41)–(43) provide a differential-equation view on iterative refinement. If we interpret each refinement step as a small update of a soft token state, then iterative refinement becomes an explicit numerical discretization of a continuous-time transport toward sharp, near-one-hot distributions. This connects to recent discrete flow matching formulations, which aim to learn continuous-time flows that can be sampled in few refinement steps while maintaining quality [Gat et al., 2024; Monsefi et al., 2025]. In this sense, diffusion modeling, flow-based modeling, and classical iterative refinement can be viewed as variants on the same theme. They differ mainly in how the intermediate state is represented (hard tokens vs. soft distributions), how the update set $\mathcal{U}$ is selected (schedule vs. confidence), and whether the dynamics are framed as a Markov process or a general learned refinement operator.

5.4.5 Diffusion Transformers

A practical reason that discrete diffusion has become competitive in NLP is that the reverse dynamics can be parameterized by Transformer. This architecture is the most scalable backbone that dominates language modeling. The term *diffusion transformer* (DiT) is most widely used in vision, where replacing the U-Net with a Transformer backbone yields strong scaling behavior [Peebles and Xie, 2023]. Presenting the details of Transformers is beyond the scope of this paper. We refer interested readers to related papers [Vaswani et al., 2017; Xiao and Zhu, 2023].

In NLP, Transformers provide a powerful mechanism for encoding a token sequence into a sequence of real-valued representations of equal length. Depending on how these representations are defined and interpreted, such models can be employed to learn arbitrary mappings from input token sequences. In diffusion modeling, Transformers must be conditioned on the diffusion time t (or a discrete index). This is typically done by injecting a learned time embedding into every layer, e.g., via additive bias, or cross-attention to a time token. The network output can be interpreted in several ways, depending on the chosen parameterization. In continuous-space diffusion, common choices include predicting the score $\nabla_{\mathbf{s}} \log p_t(\mathbf{s})$, the additive noise $\boldsymbol{\epsilon}$, the denoised data $\mathbf{s}(0)$, or a velocity variable that linearly combines the above [Ho et al., 2020; Salimans and Ho, 2022]. These parameterizations induce different discretizations and stability properties, but all can be essentially viewed as learning a time-indexed vector field that transports probability mass from a simple base distribution to the data distribution.

5.4.6 Flow-based Views

Given the modeling flexibility offered by Transformers, we need not commit to an explicit likelihood-based ELBO (see Eq. (20)) to train diffusion models. A natural alternative is flow matching. In the continuous setting, flow matching learns a time-dependent vector field whose ODE transports a simple base distribution to the data distribution. For language, the key difficulty is that the state is defined on the discrete product space V^n, so the flow must be defined either as a dynamics of probability mass over tokens (a discrete-state view), or as an ODE on a continuous relaxation

such as the probability simplex or logit space (a continuous-state view). Recent **discrete flow matching** (DFM) methods provide both perspectives and make them practical at the scale of modern language modeling [Gat et al., 2024; Campbell et al., 2024; Shaul et al., 2025].

A useful starting point is the simplex representation already introduced in Eq. (41). Instead of treating $\mathbf{x}(t)$ as a discrete token sequence, we view the intermediate state as per-position categorical probabilities

$$\mathbf{P}(t) \quad = \quad [\mathbf{p}_1(t), \ldots, \mathbf{p}_n(t)]^\top \in (\Delta^{|V|-1})^n, \tag{45}$$

where

$$\sum_{v \in V} \mathbf{p}_i(t)[v] \quad = \quad 1. \tag{46}$$

In this representation, a flow is an ODE on the product-of-simplices,

$$\frac{d\mathbf{p}_i(t)}{dt} \quad = \quad \mathbf{v}_\theta(\mathbf{p}_i(t), t, \mathbf{c}), \tag{47}$$

$$\sum_{v \in V} \mathbf{v}_\theta(\mathbf{p}_i(t), t, \mathbf{c})[v] \quad = \quad 0, \tag{48}$$

where the tangent constraint enforces probability conservation. Sampling is then an ODE integration problem which is similar to that in vision: given $\mathbf{p}(T)$ from a base distribution, integrate Eq. (48) backward to obtain $\mathbf{p}(0)$ and decode tokens by arg max or categorical sampling.

What remains is to specify the training target for $\mathbf{v}_\theta$. Discrete flow matching does this by defining a family of probability paths that interpolate between a source distribution and the data distribution, and then matching the model vector field to the instantaneous probability velocity induced by these paths. More specifically, for each training example $\mathbf{x}(0)$, one defines a conditional path $q_t(\cdot|\mathbf{x}(0), \mathbf{c})$ on V^n or on per-token marginals, and draws a noisy sample $\mathbf{x}(t) \sim q_t(\cdot|\mathbf{x}(0), \mathbf{c})$. The flow-matching regression target is the conditional velocity along that path,

$$\mathbf{v}^\star(\mathbf{x}(t), t, \mathbf{x}(0), \mathbf{c}) \quad = \quad \frac{d\mathbb{E}\big[\mathbf{1}_{\mathbf{x}(t)}|\mathbf{x}(0), \mathbf{c}\big]}{dt}, \tag{49}$$

where $\mathbf{1}_{\mathbf{x}(t)}$ is the one-hot representation for $\mathbf{x}(t)$. The resulting objective is similar to that of continuous conditional flow matching, except that the state and velocity are constrained by the simplex geometry and the categorical nature of the endpoint.

The noising kernels can be very similar to those used in the diffusion language models discussed previously. For each position, we can define a time-dependent mixture between a clean point mass and a base distribution $\bar{p}_i(\cdot)$,

$$q_t(x_i|x_i(0), \mathbf{c}) \quad = \quad \alpha_t \cdot \mathbb{I}(x_i = x_i(0)) + (1 - \alpha_t) \cdot \bar{p}_i(x_i), \tag{50}$$

with a schedule α_t decreasing from 1 to 0. This induces a smooth probability path in the simplex even though sample paths are discrete.

Differentiating Eq. (50) with respect to time yields the analytic form of the target velocity

$$\mathbf{v}_i^\star(x_i(t), t, x_i(0), \mathbf{c}) \quad = \quad \dot{\alpha}_t \cdot \left[\mathbf{1}_{x_i(0)} - \bar{p}_i\right], \tag{51}$$

where $\dot{\alpha}_t = d\alpha_t/dt$ denotes the time derivative of the schedule. This simple linear form implies that the conditional flow directs probability mass straight from the base distribution $\bar{p}_i$ towards the target token $x_i(0)$ at a rate determined by $\dot{\alpha}_t$.

To train the model, we need a conditional velocity that depends only on what is available at time t. As in standard flow matching, the training objective is defined as

$$\mathcal{L}_{\text{DFM}}(\theta) \quad = \quad \mathbb{E}_{t,\mathbf{x}(0),\mathbf{x}(t)} \left[\left\|\mathbf{v}_\theta(\mathbf{x}(t), t, \mathbf{c}) - \mathbf{v}^\star(\mathbf{x}(t), t, \mathbf{x}(0), \mathbf{c})\right\|^2\right]. \tag{52}$$

The simplex-ODE view above describes a continuous curve of marginals, but it does not specify how discrete samples should evolve during generation. A complementary view is to represent discrete flows through a CTMC parameterized by a time-dependent generator matrix $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V|\times|V|}$. The connection between the marginal velocity $\mathbf{v}(t)$ and the generator is given by the Kolmogorov forward equation

$$\frac{d\mathbf{p}_i(t)}{dt} \quad = \quad \left[\mathbf{Q}^{\text{tok}}(t)\right]^\top \mathbf{p}_i(t), \tag{53}$$

where $\mathbf{p}_i(t)$ is a vector representing the probability distribution over tokens. By expanding this matrix multiplication for a specific target token z, we can interpret the velocity as a flux balance equation

$$\mathbf{v}_i(t)[z] \quad = \quad \sum_{y\neq z} \underbrace{\mathbf{p}_i(t)[y] Q^{\text{tok}}_{yz}(t)}_{\text{inflow from } y\to z} - \sum_{y\neq z} \underbrace{\mathbf{p}_i(t)[z] Q^{\text{tok}}_{zy}(t)}_{\text{outflow from } z\to y}. \tag{54}$$

Here, $Q^{\text{tok}}_{yz}(t)$ denotes the instantaneous rate of jumping from token y to z. Note that the diagonal elements satisfy $Q^{\text{tok}}_{yy}(t) = -\sum_{z\neq y} Q^{\text{tok}}_{yz}(t)$ to ensure probability conservation.

While the marginal velocity $\mathbf{v}^\star$ derived in Eq. (51) fixes the net change in probability, it does not uniquely determine the transition rates $\mathbf{Q}^{\text{tok}}$. Discrete flow matching methods resolve this ambiguity by defining a target generator $\mathbf{Q}^{\star,\text{tok}}$ that implements the transport with minimal noise. A common choice in recent works is to define rates that only allow transitions directed towards the target sample $x_i(0)$ [Campbell et al., 2024]. Specifically, the conditional target rates are defined as

$$Q^{\star,\text{tok}}_{yz}(t \mid x_i(0)) \quad = \quad \frac{\dot{\alpha}_t}{1-\alpha_t} \cdot \mathbb{I}(z = x_i(0)), \quad \text{for } y \neq x_i(0). \tag{55}$$

This generator drives any non-target token y to jump directly to the target $x_i(0)$, while keeping the correct marginal evolution described by $\mathbf{v}^\star$.

As a result, the model is parameterized to predict this generator matrix, and the training objective becomes a rate matching loss rather than the vector regression in Eq. (52). During generation, one can simulate the process using algorithms like Gillespie's method [Gillespie, 1977]

or Tau-Leaping [Gillespie, 2001], and perform discrete jumps $x(t) \to x(t + \Delta t)$ that statistically adhere to the learned probability flow.

References

Brian D.O. Anderson. Reverse-time diffusion equation models. Stochastic Processes and their Applications, 12(3):313–326, 1982.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. Advances in neural information processing systems, 34:17981–17993, 2021.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, 19(2):263–311, 1993.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. In International Conference on Machine Learning, pages 5453–5512. PMLR, 2024.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In Proceedings of International Conference on Learning Representations, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. Advances in Neural Information Processing Systems, 37:133345–133385, 2024.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 6112–6121, 2019.

Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. The journal of physical chemistry, 81(25):2340–2361, 1977.

Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. The Journal of chemical physics, 115(4):1716–1733, 2001.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. In The Eleventh International Conference on Learning Representations, 2023.

J Gu, J Bradbury, C Xiong, VOK Li, and R Socher. Non-autoregressive neural machine translation. In International Conference on Learning Representations (ICLR), 2018.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. Advances in neural information processing systems, 32, 2019.

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

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. Advances in neural information processing systems, 34:12454–12465, 2021.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In International Conference on Learning Representations, 2017.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. Transactions of the association for computational linguistics, 8:64–77, 2020.

Hyukhun Koh, Minha Jhang, Dohyung Kim, Sangmook Lee, and Kyomin Jung. Conditional [mask] discrete diffusion language model. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 8910–8934, 2025.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1173–1182, 2018.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, 2020.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models, 2025.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. Advances in neural information processing systems, 35:4328–4343, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In Proceedings of the 41st International Conference on Machine Learning, pages 32819–32848, 2024.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In International Conference on Learning Representations, 2017.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Proceedings of the International Conference on Learning Representations (ICLR 2013), 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, pages 3111–3119, 2013b.

Amin Karimi Monsefi, Nikhil Bhendawade, Manuel Rafael Ciosici, Dominic Culver, Yizhe Zhang, and Irina Belousova. Fs-dfm: Fast and accurate long text generation with few-step diffusion language models. arXiv preprint arXiv:2509.20624, 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 preprint arXiv:2502.09992, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In Proceedings of the IEEE/CVF international conference on computer vision, pages 4195–4205, 2023.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI, 2018.

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. Journal of Machine Learning Research, 21(140):1–67, 2020.

Machel Reid, Vincent J Hellendoorn, and Graham Neubig. Diffuser: Discrete diffusion via edit-based reconstruction. arXiv preprint arXiv:2210.16886, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10684–10695, 2022.

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.

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

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In International Conference on Learning Representations, 2022.

Neta Shaul, Itai Gat, Marton Havasi, Daniel Severo, Anuroop Sriram, Peter Holderrieth, Brian Karrer, Yaron Lipman, and Ricky T. Q. Chen. Flow matching with general discrete paths: A kinetic-optimal perspective. In The Thirteenth International Conference on Learning Representations, 2025.

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.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. UL2: Unifying language learning paradigms. In The Eleventh International Conference on Learning Representations, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Tong Xiao and Jingbo Zhu. Introduction to transformers: an nlp perspective. arXiv preprint arXiv:2311.17633, 2023.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-yan Liu. A survey on non-autoregressive generation for neural machine translation and beyond. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(10):11407–11427, 2023.

Yicun Yang, Cong Wang, Shaobo Wang, Zichen Wen, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Diffusion llm with native variable generation lengths: Let [eos] lead the way, 2025.

Shuibai Zhang, Fred Zhangzhi Peng, Yiheng Zhang, Jin Pan, and Grigorios G Chrysos. Corrective diffusion language models. arXiv preprint arXiv:2512.15596, 2025.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. In International Conference on Learning Representations, 2020.