AI for Education

An Introduction to Diffusion Language Models

Ruixiang Wu, Costas Courcoubetis

The Chinese University of Hong Kong, Shenzhen

November 14, 2025

Contents

Recap: Diffusion Models

Core Idea: Diffusion models are generative models that learn to create data by reversing a gradual noising process.

This is accomplished through two opposing processes:

1. **Forward Process (Fixed)**
   - Gradually add Gaussian noise to a real image over T steps until it becomes pure noise.
   - This is a fixed, non-learned Markov process.
2. **Reverse Process (Learned)**
   - Train a neural network to gradually denoise the image, starting from pure noise, to generate a new, clean image.

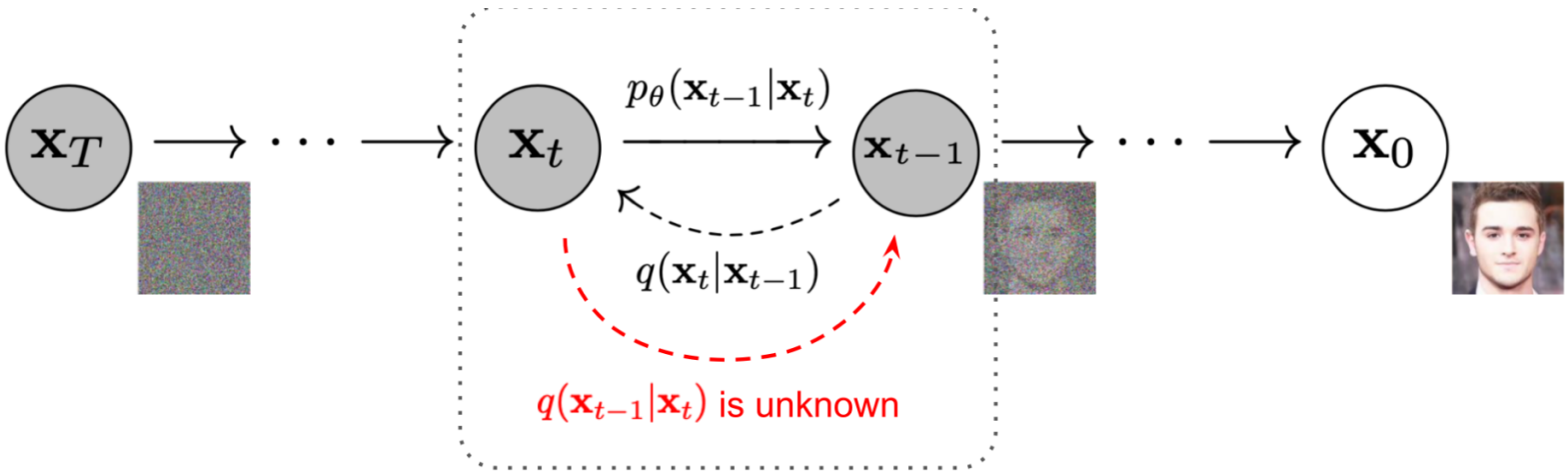
Step 1: The Forward Process (Noising)

We define a fixed Markov chain that slowly adds noise to an initial data point x_0 according to a variance schedule β_t.

$$q(\mathrm{x}_t|\mathrm{x}_{t-1}) = \mathcal{N}(\mathrm{x}_t; \sqrt{1-\beta_t}\mathrm{x}_{t-1}, \beta_t \mathrm{I})$$

- At each step t, we scale the previous image x_{t-1} and add some noise.
- As t increases, the original data signal fades.
- When T is large enough, x_T is indistinguishable from pure isotropic Gaussian noise, $\mathrm{x}_T \sim \mathcal{N}(0, \mathrm{I})$.

$$\mathrm{x}_0 \xrightarrow{+\text{noise}} \mathrm{x}_1 \xrightarrow{+\text{noise}} \cdots \xrightarrow{+\text{noise}} \mathrm{x}_T$$



Step 2: The Reverse Process (Denoising)

To generate data, we must learn to reverse the forward process. This means estimating the conditional probability $p(\mathrm{x}_{t-1}|\mathrm{x}_t)$.

- The true reverse distribution $q(\mathrm{x}_{t-1}|\mathrm{x}_t)$ is intractable to calculate.
- **Solution:** We train a neural network p_θ to approximate it. The network's job is to predict the parameters of the distribution for the previous, less noisy step.

$$p_\theta(\mathrm{x}_{t-1}|\mathrm{x}_t) = \mathcal{N}(\mathrm{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathrm{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathrm{x}_t, t))$$

- The function $\boldsymbol{\mu}_\theta(\mathrm{x}_t, t)$ is the neural network. It takes the current noisy image x_t and the timestep t as input and predicts the mean $\boldsymbol{\mu}$ of the slightly denoised image x_{t-1}.
- In practice, the network is often trained to simply predict the **noise** that was added at step t.

$$\mathrm{x}_0 \xleftarrow{\text{denoise}} \mathrm{x}_1 \xleftarrow{\text{denoise}} \cdots \xleftarrow{\text{denoise}} \mathrm{x}_T \sim \mathcal{N}(0, \mathrm{I})$$

Mathematical Foundations (1) I

Goal: Maximize the probability of the reverse process: $p_\theta(x_0)$.
Notations:

$$p_\theta(x_{0:T}) \coloneqq p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \quad p_\theta(x_{t-1}|x_t) \coloneqq \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

$$q(x_{1:T}|x_0) \coloneqq \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) \coloneqq \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathrm{I})$$

Training is performed by optimizing the 'variational bound on negative log likelihood:

$$\begin{aligned} \mathbb{E}[-\log p_\theta(x_0)] \;&\leq\; \mathbb{E}_q[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}] \\ &= \mathbb{E}_q\Big[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \Big] =: L, \end{aligned}$$

Mathematical Foundations (1) II

We can rewritten L as:

$$\mathbb{E}_q \Bigg[\underbrace{\mathsf{D}_{KL}(q(x_T|x_0)\|p(x_T))}_{L_T} + \sum_{t>1} \underbrace{\mathsf{D}_{KL}(q(x_{t-1}|x_t,x_0)\|p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{-\log p_\theta(x_0|x_1)}_{L_0} \Bigg]$$

where:

$$q(x_{t-1}|x_t,x_0) = \mathcal{N}(x_{t-1};\tilde{\mu}_t(x_t,x_0),\tilde{\beta}_t\mathrm{I}), p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\mu_\theta(x_t,t),\sigma_t^2\mathrm{I})$$

$$\tilde{\mu}_t(x_t,x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

We can further rewrite the term L_{t-1} as:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(x_t,x_0) - \mu_\theta(x_t,t)\|^2 \right] + C$$

Contents

Comparison: AR-LMs and DLMs I

Autoregressive Language Models:

- Most well-known language models today are **autoregressive** (AR).
  - Examples: GPT-2, Llama, Gemini, ChatGPT, Claude.
  - They use the Transformer backbone for decoding.
- **Core Idea:** They predict one token at a time, from left to right.
  - The prediction of the next token depends on the sequence of previous tokens.
  - This is a sequential, token-by-token generation process.

Diffusion Language Models:

- DLMs work differently. They are **non-autoregressive** (NAR).
- Instead of predicting token-by-token, they **iteratively refine** the *whole sequence* from a noisy starting point.
- **The Analogy:** Think of it as starting with a messy, nonsensical paragraph (random noise) and repeatedly cleaning it up until it becomes a polished, coherent passage.

Comparison: AR-LMs and DLMs II

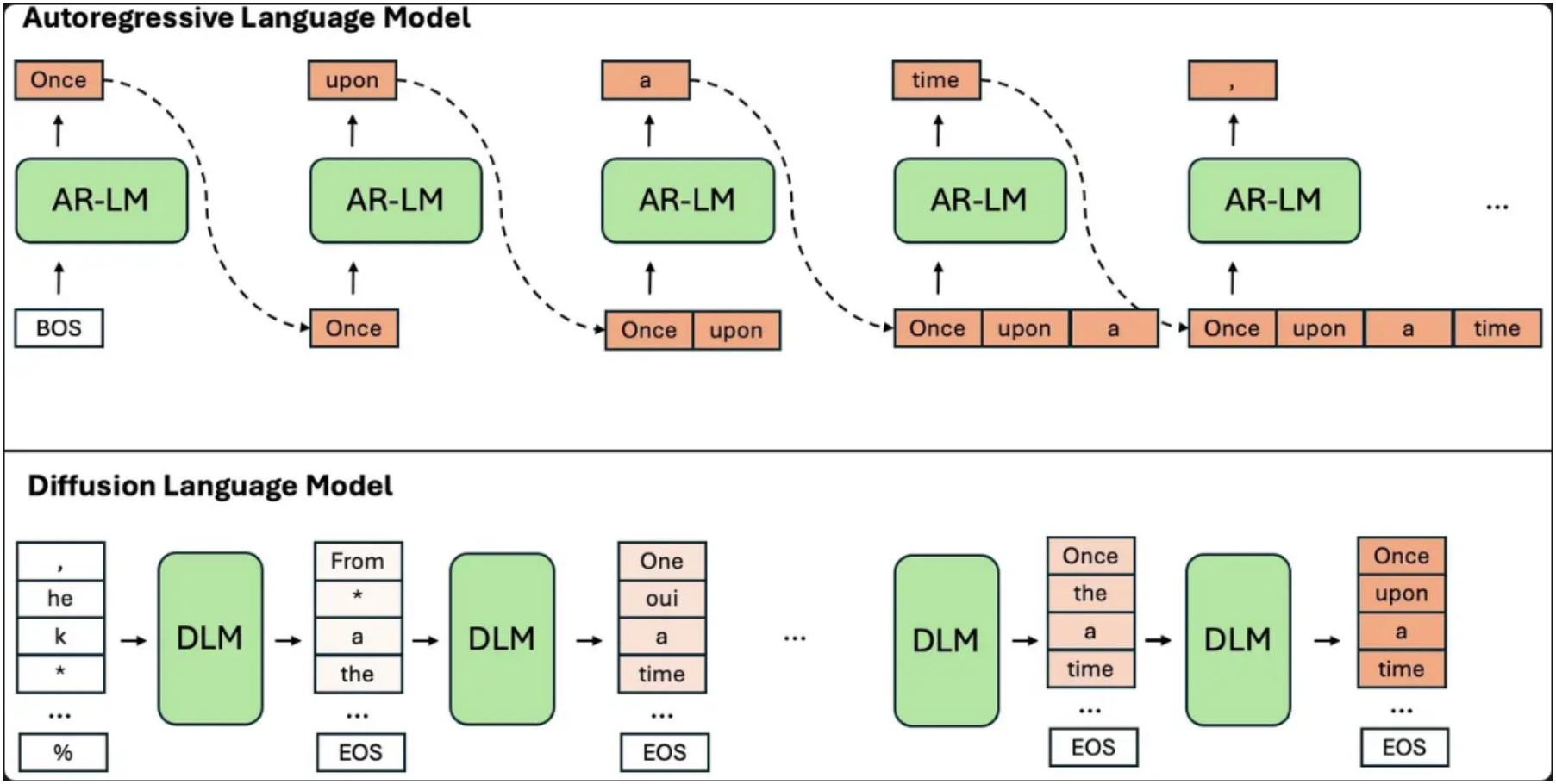


Figure: Visual Comparison Between AR-LMs and DLMs

Comparison: AR-LMs and DLMs III

For a sequence $x = \{x_1, x_2, \ldots, x_N\}$:

An AR-LM models the probability as a product of conditional probabilities:

$$P(x;\theta) = \prod_{n=1}^{N} P(x_n | x_{<n}; \theta) \tag{1}$$

A DLM models a different kind of distribution:

- It learns a *reverse diffusion process* that evolves over time t.
- This process transforms a sequence from pure noise (x_T) back to clean text (x_0).

$$x_{t-1} \sim p_\theta(x_{t-1} | x_t, t) \tag{2}$$

Advantages of Diffusion Language Models

1. Non-Autoregressive Generation: Since sequences are generated holistically, the model can fix earlier mistakes as it refines the output.

- This overcomes the **error propagation** problem inherent in sequential AR models.

2. Superior Controllability: DLMs are naturally well-suited for controllable generation, offering fine-grained control over the output.

- **Length:** Precisely control the length of the generated passage.
- **Content:** Easily perform tasks like text edits and infilling.
- **Structure:** Enforce constraints for generating code, tables, etc.
- **Style:** Steer the output style using methods like adding a classifier.

Challenges?:

Advantages of Diffusion Language Models

1. Non-Autoregressive Generation: Since sequences are generated holistically, the model can fix earlier mistakes as it refines the output.

- This overcomes the **error propagation** problem inherent in sequential AR models.

2. Superior Controllability: DLMs are naturally well-suited for controllable generation, offering fine-grained control over the output.

- **Length:** Precisely control the length of the generated passage.
- **Content:** Easily perform tasks like text edits and infilling.
- **Structure:** Enforce constraints for generating code, tables, etc.
- **Style:** Steer the output style using methods like adding a classifier.

Challenges?: there's a big discrepancy between traditional continuous diffusion models (which crushed it in image generation) and the world of discrete text.

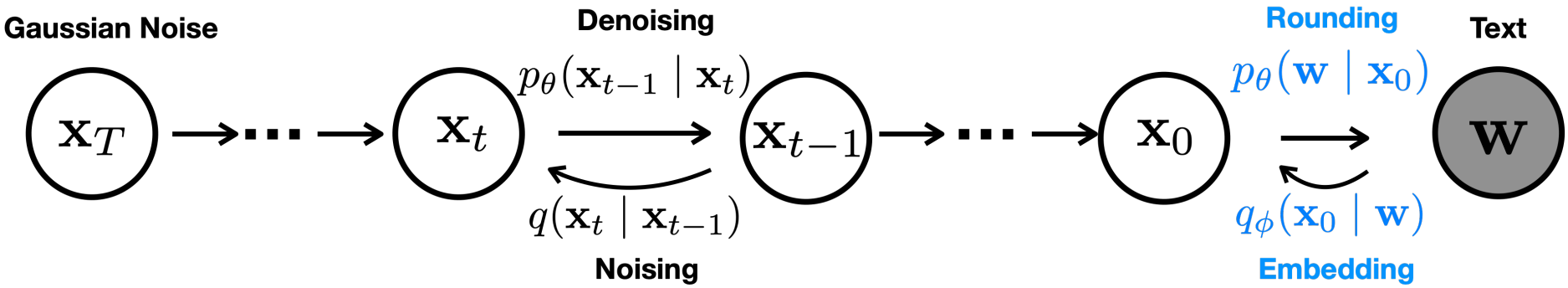
Solution 1: Adding Noise to the Embedding of Tokens I

Diffusion-LM Improves Controllable Text Generation (2)

The Idea: End-to-End Trainable Embeddings

Instead of using fixed, pre-trained word embeddings, the paper proposes learning the embeddings **together** with the diffusion model itself.

- An embedding function $\text{EMB}(\text{w})$ maps a sequence of words to a sequence of vectors.
- The model's training objective is modified to learn both the diffusion process parameters and the word embedding parameters simultaneously.



Solution 1: Adding Noise to the Embedding of Tokens II

Training Objective for standard diffusion loss:

$$\mathcal{L}_{\text{vlb}}(\mathrm{x}_0) = \mathop{\mathbb{E}}_{q(\mathrm{x}_{1:T}|\mathrm{x}_0)} \left[\log \frac{q(\mathrm{x}_T|\mathrm{x}_0)}{p_\theta(\mathrm{x}_T)} + \sum_{t=2}^{T} \log \frac{q(\mathrm{x}_{t-1}|\mathrm{x}_0, \mathrm{x}_t)}{p_\theta(\mathrm{x}_{t-1}|\mathrm{x}_t)} - \log p_\theta(\mathrm{x}_0|\mathrm{x}_1) \right]$$

Adding terms for mapping to and from the word-embedding space:

$$\mathcal{L}^{\text{e2e}}_{\text{vlb}}(\mathrm{w}) = \mathop{\mathbb{E}}_{q_\phi(\mathrm{x}_0|\mathrm{w})} \left[\mathcal{L}_{\text{vlb}}(\mathrm{x}_0) + \log q_\phi(\mathrm{x}_0|\mathrm{w}) - \log p_\theta(\mathrm{w}|\mathrm{x}_0) \right]],$$

$$\mathcal{L}^{\text{e2e}}_{\text{simple}}(\mathrm{w}) = \mathop{\mathbb{E}}_{q_\phi(\mathrm{x}_{0:T}|\mathrm{w})} \left[\mathcal{L}_{\text{simple}}(\mathrm{x}_0) + ||\textsc{Emb}(\mathrm{w}) - \mu_\theta(\mathrm{x}_1, 1)||^2 - \log p_\theta(\mathrm{w}|\mathrm{x}_0) \right].$$

Challenges?:

Solution 1: Adding Noise to the Embedding of Tokens III

Training Objective for standard diffusion loss:

$$\mathcal{L}_{\text{vlb}}(\mathrm{x}_0) = \mathop{\mathbb{E}}_{q(\mathrm{x}_{1:T}|\mathrm{x}_0)} \left[\log \frac{q(\mathrm{x}_T|\mathrm{x}_0)}{p_\theta(\mathrm{x}_T)} + \sum_{t=2}^{T} \log \frac{q(\mathrm{x}_{t-1}|\mathrm{x}_0, \mathrm{x}_t)}{p_\theta(\mathrm{x}_{t-1}|\mathrm{x}_t)} - \log p_\theta(\mathrm{x}_0|\mathrm{x}_1) \right]$$

Adding terms for mapping to and from the word-embedding space:

$$\mathcal{L}^{\text{e2e}}_{\text{vlb}}(\mathrm{w}) = \mathop{\mathbb{E}}_{q_\phi(\mathrm{x}_0|\mathrm{w})} \left[\mathcal{L}_{\text{vlb}}(\mathrm{x}_0) + \log q_\phi(\mathrm{x}_0|\mathrm{w}) - \log p_\theta(\mathrm{w}|\mathrm{x}_0) \right]],$$

$$\mathcal{L}^{\text{e2e}}_{\text{simple}}(\mathrm{w}) = \mathop{\mathbb{E}}_{q_\phi(\mathrm{x}_{0:T}|\mathrm{w})} \left[\mathcal{L}_{\text{simple}}(\mathrm{x}_0) + ||\mathrm{EMB}(\mathrm{w}) - \mu_\theta(\mathrm{x}_1, 1)||^2 - \log p_\theta(\mathrm{w}|\mathrm{x}_0) \right].$$

Challenges?: The reverse process requires "rounding" a final generated vector x_0 to the nearest word embedding in the vocabulary. Empirically, the model often generates a final vector x_0 that doesn't perfectly align with any single word embedding. This leads to rounding errors and affects generation quality.

Solution 1: Adding Noise to the Embedding of Tokens IV

Noticed that:

$$
\begin{aligned}
&||\hat{\mu}(\mathrm{x}_t, \mathrm{x}_0) - \mu_\theta(\mathrm{x}_t, t)||^2 \\
=&||(\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathrm{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathrm{x}_t) - (\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} f_\theta(\mathrm{x}_t, t) + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathrm{x}_t)||^2 \\
=&||\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}(\mathrm{x}_0 - f_\theta(\mathrm{x}_t, t))||^2 \propto ||\mathrm{x}_0 - f_\theta(\mathrm{x}_t, t)||^2
\end{aligned}
$$

Diffusion-LM's Approach

Re-parametrize the objective so the model **directly predicts the final clean output** x_0 from any noisy input x_t.

$$
\mathcal{L}_{\text{simple}} \propto \sum_{t=1}^{T} \mathbb{E}_{\mathrm{x}_t} ||f_\theta(\mathrm{x}_t, t) - \mathrm{x}_0||^2
$$

This forces the model to think about the discrete target at every single step.

Solution 1: Adding Noise to the Embedding of Tokens V

To further reduce rounding errors during generation, a trick is applied at each denoising step.

1. At any step t, the model first predicts its best guess for the final, clean output: $f_\theta(\mathrm{x}_t, t)$.
2. **The Clamp:** Before proceeding, this predicted vector is mapped to its *nearest neighbor* in the actual word embedding vocabulary.

$$f_\theta(\mathrm{x}_t, t) \to \mathrm{Clamp}(f_\theta(\mathrm{x}_t, t))$$

3. This "clamped," valid embedding is then used to compute the next, less noisy state x_{t-1}.

The Intuition

This trick forces the model to "commit" to a sequence of valid words throughout the entire generation process, making the final prediction much more precise.

Solution 1: Adding Noise to the Embedding of Tokens VI

The Classic "Plug-and-Play" Approach

This is typically done by sampling from a posterior distribution using Bayes' rule:

$$p(\mathrm{w} \mid \mathrm{c}) \propto \underbrace{p_{\mathsf{lm}}(\mathrm{w})}_{\text{Fluency}} \cdot \underbrace{p(\mathrm{c} \mid \mathrm{w})}_{\text{Control}}$$

- A pre-trained **Language Model** ($p_{\mathsf{lm}}(\mathrm{w})$) ensures the output text is fluent.
- A separate **Classifier** ($p(\mathrm{c} \mid \mathrm{w})$) scores how well the text satisfies the control.

Applying Bayes' Rule to the Latent Variable

The desired posterior is approximated in the continuous space:

$$p(\mathrm{x}_{t-1} \mid \mathrm{x}_t, \mathrm{c}) \propto \underbrace{p(\mathrm{x}_{t-1} \mid \mathrm{x}_t)}_{\text{Fluency (from Diffusion-LM)}} \cdot \underbrace{p(\mathrm{c} \mid \mathrm{x}_{t-1})}_{\text{Control (from a Classifier)}}$$

Solution 2: Discrete Diffusion over Tokens I

Large Language Diffusion Models (3) (an 8B-parameter DLM that is trending.)

LLaDA's Approach: Iterative Refinement

LLaDA generates text by starting with a fully masked sequence and iteratively predicting **all missing tokens simultaneously** until the text is complete.

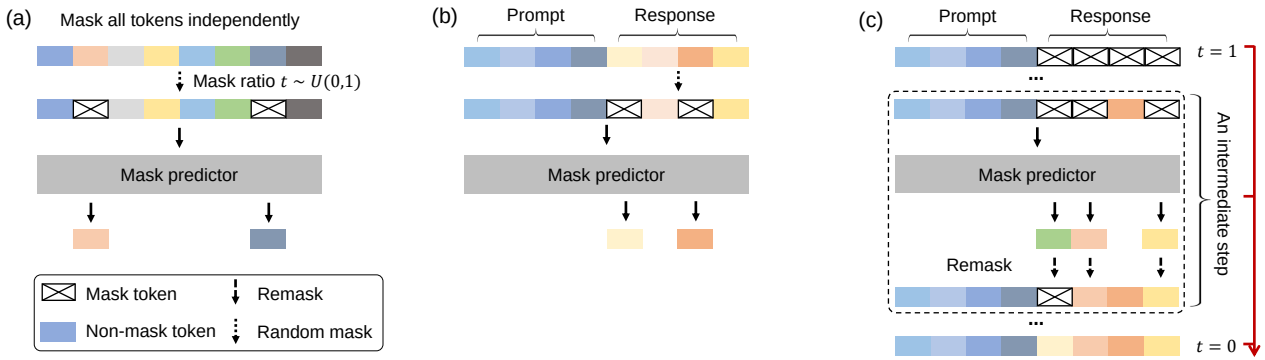


Figure: **A Conceptual Overview of LLaDA.** (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked) to $t = 0$ (unmasked), predicting all masks simultaneously at each step with flexible remask strategies.

Solution 2: Discrete Diffusion over Tokens II

The Forward Process The "corruption" of a clean sequence x_0 into a masked sequence x_t is controlled by a continuous time variable $t \in [0, 1]$, which represents the masking probability.

Mathematical Formulation of Masking

For each token x_0^i in the original sequence x_0, the corresponding token x_t^i in the masked sequence is determined as follows:

$$x_t^i = \begin{cases} \texttt{[MASK]} & \text{with probability } t \\ x_0^i & \text{with probability } 1 - t \end{cases}$$

- When $t = 0$, the sequence is completely clean (x_0).
- When $t = 1$, the sequence is fully masked.
- For training, t is sampled uniformly from $[0, 1]$ for each data sample.

Solution 2: Discrete Diffusion over Tokens III

The Reverse Process: The core of LLaDA is a **mask predictor**, $p_\theta(\cdot|x_t)$, a Transformer model trained to predict all masked tokens at once.

LLaDA's Loss Function

The model is trained using a cross-entropy loss computed only on the masked tokens. The objective is to minimize:

$$\mathcal{L}(\theta) = -\mathbb{E}_{t,x_0,x_t}\left[\underbrace{\frac{1}{t}}_{\text{Scaling Factor}}\sum_{i=1}^{L}\underbrace{1[x_t^i = \mathtt{M}]}_{\text{Only for masked tokens}}\underbrace{\log p_\theta(x_0^i|x_t)}_{\text{Cross-Entropy Loss}}\right]$$

Solution 2: Discrete Diffusion over Tokens IV

Other types of discrete diffusion over tokens:

- Using "Softmax" idea (4; 5; 6):

$$q(\mathrm{x_t}|\mathrm{x_{t-1}}) = \mathsf{Categorize}(\mathrm{x_t}; \mathrm{p} = \mathrm{x_{t-1}}\mathrm{Q_t})$$

- Token Substitution: Randomly replacing tokens with other tokens from the vocabulary (7).
- Text as Image: Instead of dealing with the discrete gap, rendering the target text as glyph images containing visual language content (8).

Contents

References I

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[2] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto, "Diffusion-lm improves controllable text generation," 2022.

[3] S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li, "Large language diffusion models," 2025.

[4] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, "Argmax flows and multinomial diffusion: Learning categorical distributions," 2021.

[5] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," 2023.

[6] A. Lou, C. Meng, and S. Ermon, "Discrete diffusion modeling by estimating the ratios of the data distribution," 2024.

[7] H. Zou, Z. M. Kim, and D. Kang, "A survey of diffusion models in natural language processing," 2023.

[8] J. Li, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, "Glyphdiffusion: Text generation as image generation," 2023.