

Cola2: Co-Adaptive Hierarchical Language Modeling with Continuous Semantic Compression and Balanced Discrete Generation

Hongcan Guo^{1,2,†}, Yian Zhao^{2,3}, Feng Wang², Shen Nie⁴, Guoqiang Wei^{2,†},
Yan Zeng^{2,†,‡}, Hengshuang Zhao^{1,✉}

¹The University of Hong Kong, ²ByteDance Seed, ³Peking University,
⁴Renmin University of China

[†]Project Lead, [‡]Major Contributor, [✉]Corresponding author

Abstract

Hierarchical language generation separates continuous semantic modeling from discrete text realization, but its effectiveness depends on what information the latent space should compress and how the discrete generator should use it. We propose *Cola2*, which jointly studies continuous semantic compression, the discrete reconstruction-generation balance, and their co-adaptation. *Cola2* dynamically segments text into semantic spans, controls the compression rate with an auxiliary-loss-free negative-feedback mechanism, and combines reconstruction and generation supervision to learn compact and predictive latent representations. A block-causal masked diffusion model (MDM) decoder is jointly trained under clean, noisy, and dropped latent conditions, and classifier-free guidance (CFG) adjusts the balance between semantic reconstruction and autonomous generation. During training, token-prediction gradients update the encoder through the latents, driving the continuous representation and discrete generation to co-adapt, while Flow Matching prior learning is decoupled from representation learning via stop-gradient. At inference, the generated text is re-encoded under the fixed span layout, keeping the latent history consistent with the actual text context. Systematic component analyses and controlled comparisons on five downstream tasks with about 210B training tokens validate the core designs and show competitive, unsaturated scaling. With only 346.03B source tokens, the unconverged checkpoint reaches a 60.10 eight-task average and surpasses Llama-3-8B on HellaSwag and StoryCloze despite its over 43× larger reported pretraining budget. These results indicate effective transfer of the Qwen-initialized causal decoder’s generation capability and substantial scaling headroom for hierarchical language generation. More broadly, continuous-latent hierarchical modeling can combine effectively with established token-space generation paradigms, including autoregressive and masked-diffusion decoders.

Date: October 22, 2026

Project Lead: Yan Zeng at zengyan.yanne@bytedance.com

Correspondence: Hengshuang Zhao at hszhao@cs.hku.hk

Project Page: <https://hongcanguo.github.io/Cola2>

★ Seedance Team Research Project ★

1 Introduction

Large language models (LLMs) have made remarkable progress under the autoregressive (AR) paradigm [6, 43], but the fixed left-to-right generation order imposes an inherent serial dependency on inference. Diffusion language models (DLMs) [1, 2, 34, 41] offer a new way to break this constraint. Cola DLM [13] further decomposes text generation into semantic organization in a continuous latent space and text realization in a discrete space; through latent prior modeling and conditional decoding, it provides a viable path toward hierarchical language generation beyond direct token modeling.

However, a structural division between continuous and discrete spaces does not automatically yield an effective division of labor. The key questions are what information the latent space should compress and how the discrete generator should use it. Shortening the latent sequence is not equivalent to obtaining a compact semantic representation, and a reconstruction-only objective does not guarantee that the latents can predict missing or future text [19, 24, 29]. Meanwhile, the decoder must not only faithfully realize the latent semantics, but also generate the remaining content from the text context when the latent condition is incomplete or imprecise [44]. Semantic compression and discrete generation therefore cannot be designed independently: the information the latents should retain must be determined jointly with the information the decoder can complete.

This interdependence further raises the question of how continuous and discrete representations should be learned jointly. Fixing the encoder in advance determines the information carried by the latents before the discrete generator is fully learned, limiting the continual adaptation of the representation to decoding needs. At the same time, letting the latent-prior fitting objective directly shape the encoder may push the latents toward directions that are easier to fit but not necessarily more beneficial for discrete generation. Hierarchical language modeling thus requires not only learning continuous representations and a discrete generator, but also specifying which objectives should jointly shape the representation and which learning processes should remain decoupled.

To address these issues, we propose *Cola2*, which jointly studies continuous semantic compression, the discrete reconstruction-generation balance, and their co-adaptation within a unified hierarchical language modeling framework. Its core idea is that, under limited latent capacity, the actual demands of text reconstruction and generation jointly determine what the continuous representation should retain, while the discrete generator handles the residual uncertainty that the latents do not explain. Specifically, *Cola2* dynamically segments text into semantic spans, controls the compression rate with an auxiliary-loss-free negative-feedback mechanism, and combines reconstruction and generation supervision to learn compact and predictive latent representations. On the discrete side, a block-causal masked diffusion model (MDM) decoder [1, 41] is jointly trained under clean, noisy, and dropped latent conditions, balancing semantic reconstruction with context-based autonomous generation, and classifier-free guidance (CFG) [15] adjusts this balance at inference.

During training, the token-prediction loss is backpropagated through the latents to the encoder, so that the continuous representation and discrete generation co-adapt; the Diffusion Transformer (DiT) [38] learns the current latent distribution via Flow Matching [26] without backpropagating gradients to the encoder, thereby decoupling prior fitting from representation learning. At inference, the model re-encodes the generated text block by block under the fixed span layout, keeping the latent history consistent with the actual text context and aligning the two spaces throughout generation.

Building on these ideas, we systematically study co-adaptive hierarchical language generation across continuous and discrete spaces from three perspectives: probabilistic modeling, representation learning, and generation performance. Our main contributions are as follows:

- **We propose *Cola2*, a co-adaptive hierarchical language modeling framework.** From the perspective of conditional information decomposition, we analyze semantic compression, the reconstruction-generation balance, and continuous-discrete co-adaptation in a unified manner, clarifying the relation between the information retained by the latents and the residual left to discrete generation.
- **We develop a generation-oriented joint learning mechanism for continuous and discrete representations.** It combines dynamic semantic compression, generation-oriented supervision, and training under latent

conditions of varying reliability, and coordinates representation learning with prior fitting through gradient decoupling, so that compact semantic representations and discrete generation capability co-evolve.

- **We validate the core designs and their scaling potential through systematic experiments.** Controlled five-task scaling to about 210B tokens and a separate 346.03B-source-token pretraining run show competitive, unsaturated scaling. The latter reaches a 60.10 average over eight tasks and surpasses Llama-3-8B on HellaSwag and StoryCloze despite over $43\times$ fewer reported tokens, demonstrating decoder capability transfer and compatibility with autoregressive and masked-diffusion token-space generators.

2 Related Work

2.1 Semantic Compression and Hierarchical Language Models

Fixed-granularity methods shorten the modeled sequence through chunking or downsampling [33, 48]. Adaptive compression methods dynamically aggregate bytes or tokens according to content [18, 36, 39]. Concept-level methods organize generation through sentence representations or concept prediction [3, 7]. In contrast, *Cola2* maintains the target compression rate through negative-feedback control without an auxiliary compression-rate loss, and combines reconstruction and generation supervision to learn compact and predictive continuous semantic representations.

2.2 Token-Space Language Models

Autoregressive methods generate text serially, token by token [6, 11, 45]. Discrete diffusion methods predict multiple positions in parallel through iterative denoising [2, 27, 34, 41, 42, 47], and have been further extended to block-level generation and token editing [1, 4, 5]. Continuous-augmented methods use paired continuous states to assist discrete denoising [44, 51, 52]. In contrast, *Cola2* learns a semantic prior in a compressed latent space and then performs conditional generation with a discrete decoder, handling semantic modeling and text realization hierarchically.

2.3 Continuous Representations and Latent-Space Language Models

Token-aligned methods perform diffusion or flow modeling on continuous representations [8, 10, 12, 16, 22, 23, 46]. Autoencoder-based latent-space methods separate latent generation from text decoding [28, 29, 50]. Joint-learning methods further co-optimize the latent space and the generative model [13, 30]. In contrast, *Cola2* drives representation learning with discrete reconstruction and generation supervision while isolating the prior-fitting gradient, so that semantic compression and the decoder’s reconstruction-generation balance are optimized jointly.

3 Cola2: Continuous Latent Diffusion for Hierarchical Text Modeling

We formulate hierarchical text generation as a continuous latent-variable model and, from an information-decomposition perspective, discuss semantic compression, the reconstruction-generation balance, and continuous-discrete co-adaptation (Section 3.1). We then present the unified model, training procedure, latent-feedback inference procedure, and computational cost (Sections 3.2–3.3).

3.1 Probabilistic and Information-Theoretic Perspective

3.1.1 Hierarchical Probabilistic Modeling

Let C denote the observed context, Y the text to be generated, and $Z = (z_1, \dots, z_K)$ a continuous latent sequence with $z_k \in \mathbb{R}^{d_z}$. If Y contains T tokens, the compression rate is $\rho = K/T$. *Cola2* factorizes the text probability into a continuous latent prior and discrete conditional generation:

$$p(Y | C) = \int p_\theta(Z | C) p_\psi(Y | C, Z) dZ. \quad (3.1)$$

During training, introducing the encoder $q_\phi(Z | C, Y)$ as an approximate posterior yields the conditional evidence lower bound (ELBO)

$$\begin{aligned} \mathcal{L}_{\text{cELBO}} &= \mathbb{E}_{q_\phi} [\log p_\psi(Y | C, Z)] \\ &\quad - \text{KL}(q_\phi(Z | C, Y) \| p_\theta(Z | C)) \leq \log p(Y | C). \end{aligned} \quad (3.2)$$

Defining the conditional aggregated posterior

$$\bar{q}_\phi(Z | C) = \int q_\phi(Z | C, Y) p_{\text{data}}(Y | C) dY, \quad (3.3)$$

the average negative ELBO decomposes as

$$\begin{aligned} -\mathbb{E} \mathcal{L}_{\text{cELBO}} &= \mathbb{E} [-\log p_\psi(Y | C, Z)] + I(Y; Z | C) \\ &\quad + \mathbb{E}_C \text{KL}(\bar{q}_\phi(Z | C) \| p_\theta(Z | C)). \end{aligned} \quad (3.4)$$

Thus, the encoder determines which information is retained within the limited latent capacity, the discrete generator decoder implements $p_\psi(Y | C, Z)$, and the latent DiT only needs to fit the $p_\theta(Z | C)$ induced by the online encoder.

3.1.2 Semantic Compression and Information Density

Given C , the uncertainty of the target text satisfies

$$H(Y | C) = I(Y; Z | C) + H(Y | C, Z). \quad (3.5)$$

If S denotes a high-level semantic variable determined by Y , the chain rule of mutual information gives

$$\begin{aligned} H(Y | C) &= \underbrace{I(S; Z | C)}_{\text{semantic information}} + \underbrace{I(Y; Z | S, C)}_{\text{local residual information}} \\ &\quad + \underbrace{H(Y | C, Z)}_{\text{discrete generation residual}}. \end{aligned} \quad (3.6)$$

Under the compression constraint $K = \rho T$, whether the latents are worth further modeling in continuous space depends not only on whether K is sufficiently short, but more importantly on whether the information retained in the limited latent positions is semantic and predictive. To characterize this, we define the analytical quantity

$$\mathcal{D}_{\text{sem}} := \frac{I(S; Z | C)}{K} = \frac{I(S; Z | C)}{\rho T}, \quad (3.7)$$

which measures the density of high-level semantic information carried per latent position. This quantity characterizes the design objective rather than serving as a loss estimated directly during training.

Furthermore, partitioning Y into a visible part Y^{vis} and a target part Y^{tar} gives

$$\begin{aligned} I(Y^{\text{tar}}; Z | C, Y^{\text{vis}}) &= H(Y^{\text{tar}} | C, Y^{\text{vis}}) \\ &\quad - H(Y^{\text{tar}} | C, Y^{\text{vis}}, Z). \end{aligned} \quad (3.8)$$

Therefore, beyond local reconstructability, a high-quality latent space should also reduce the conditional uncertainty of future or missing tokens, i.e., concentrate more generative information within its limited capacity. Section 3.2 instantiates this principle through dynamic semantic compression and generation-oriented latent supervision.

3.1.3 Reconstruction--Generation Balance and Continuous--Discrete Co-adaptation

Let $\tilde{Z}$ be the latent condition obtained by passing Z through a stochastic degradation channel, satisfying the conditional Markov chain $Y \rightarrow Z \rightarrow \tilde{Z}$ given C . By the data processing inequality,

$$I(Y; Z | C) \geq I(Y; \tilde{Z} | C), \quad H(Y | C, Z) \leq H(Y | C, \tilde{Z}). \quad (3.9)$$

When the latent condition is intact, the decoder’s main task is to realize the continuous semantics as the corresponding discrete tokens; as the condition is perturbed or removed, $H(Y | C, \tilde{Z})$ increases, and more uncertainty must be handled by the decoder’s own discrete prediction capability. This gives an information-theoretic form of the reconstruction–generation trade-off.

On the other hand, since $H(Y | C)$ is determined by the data distribution, changing the encoder effectively reallocates

$$H(Y | C) = I(Y; Z_\phi | C) + H(Y | C, Z_\phi). \quad (3.10)$$

If E_ϕ is fixed before the decoder is trained, this allocation is determined before the discrete generator is learned. *Cola2* instead jointly optimizes

$$(\phi^*, \psi^*) = \arg \min_{\phi, \psi} \mathbb{E}_{p_{\text{data}}(C, Y)} \mathbb{E}_{q_\psi(Z | C, Y)} [-\log p_\psi(Y | C, Z)] \quad \text{s.t.} \quad \frac{K}{T} \approx \rho^*, \quad (3.11)$$

so that token prediction retains a direct gradient to the encoder:

$$\nabla_\phi \mathcal{L}_{\text{tok}} = \frac{\partial \mathcal{L}_{\text{tok}}}{\partial Z} \frac{\partial Z}{\partial \phi} \neq 0. \quad (3.12)$$

This lets the continuous latent space and the discrete realization co-adapt: the encoder learns semantic representations better suited to reconstruction and generation by the actual decoder, while latent-prior fitting is decoupled from this representation learning. The concrete mechanisms for compression, reliability training, gradient routing, and inference alignment are given in the next section.

3.2 Workflow

Figure 1 instantiates the above information decomposition with three coordinated modules: the compression variational autoencoder (VAE) encoder E_ϕ learns continuous latents, the discrete generator decoder D_ψ realizes the latent condition as discrete tokens, and the block-causal DiT G_θ learns the generative distribution of the detached online latents. Their relationship can be summarized as

$$(C, Y) \xrightarrow{E_\phi} Z, \quad (C, Z) \xrightarrow{D_\psi} Y, \quad \text{sg}(Z) \xrightarrow{\mathcal{L}_{\text{FM}}} G_\theta. \quad (3.13)$$

Here D_ψ is a *block-causal masked-diffusion decoder* (MDM), which is causal across blocks and performs bidirectional token denoising within the current block. Stage 1 jointly learns E_ϕ and D_ψ ; Stage 2 retains this pathway and additionally trains a detached latent DiT. Full implementation details are given in Appendix G.

3.2.1 Model Architecture

Compression VAE encoder. The target sequence $Y = (y_1, \dots, y_T)$ is dynamically partitioned into K contiguous spans. Let $\mathcal{S}_k$ denote the token index set of the k -th span and $L_k = |\mathcal{S}_k|$ its length. Then

$$\bigcup_{k=1}^K \mathcal{S}_k = \{1, \dots, T\}, \quad 1 \leq L_k \leq L_{\max}, \quad \rho = \frac{K}{T}. \quad (3.14)$$

The encoder E_ϕ first performs dynamic segmentation over token representations, then aggregates bidirectionally within each span and models causally along the latent axis, yielding a diagonal Gaussian posterior

$$q_\phi(z_k | C, Y) = \mathcal{N}(\mu_k, \text{diag}(\sigma_k^2)), \quad z_k = \mu_k + \sigma_k \odot \epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, I). \quad (3.15)$$

The compression rate is maintained near the target ρ^* by an independent, gradient-free rate-bias controller rather than learned through an additional differentiable rate loss. The dynamic boundaries, span-local pooling, and convergence properties of the controller are given in Appendix G.1.

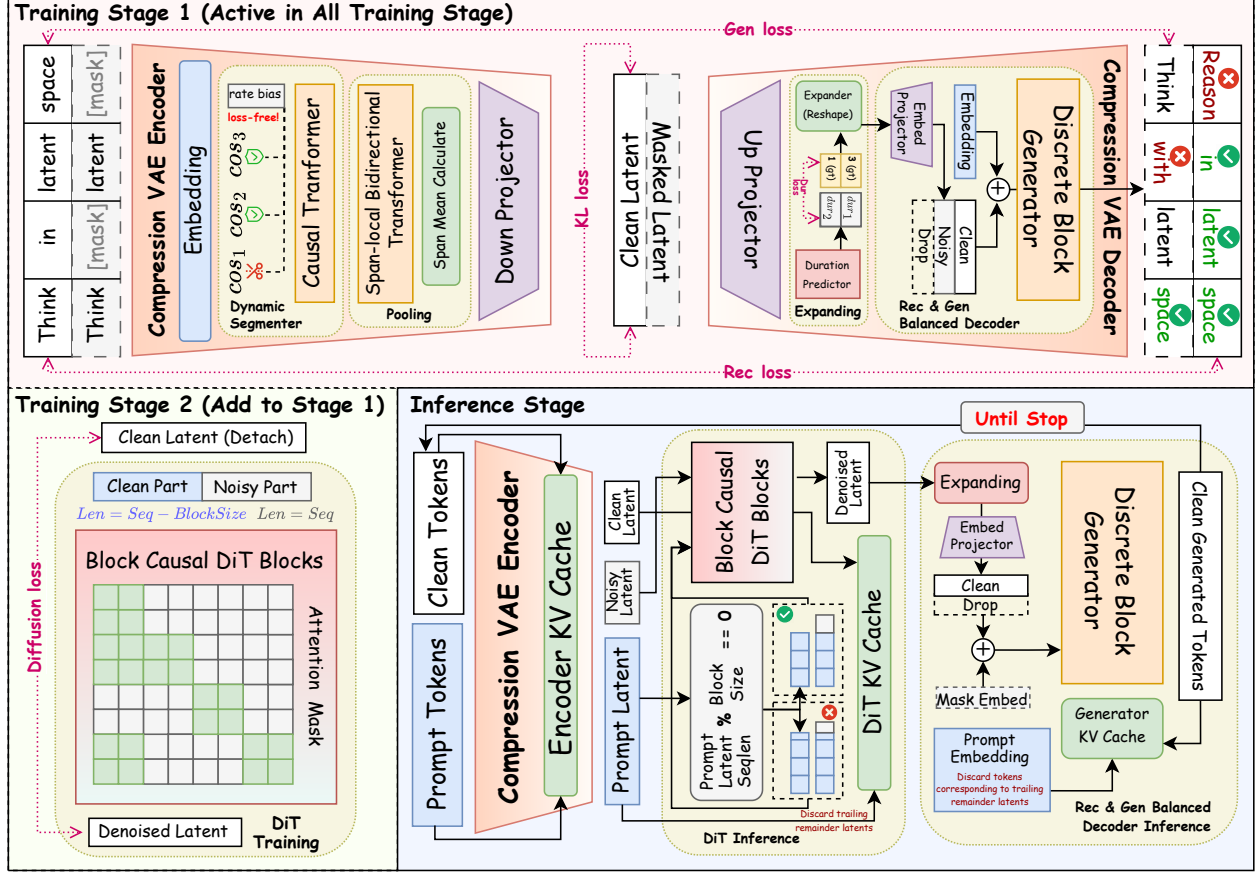


Figure 1 Method overview. Stage 1 jointly trains the compression encoder and the reconstruction-generation balanced decoder; Stage 2 additionally trains a DiT on detached latents; at inference, the DiT generates latents block by block and the decoder decodes them into tokens.

Latent-to-token interface and block-causal MDM decoder. Each latent is first projected into the decoder hidden space, and a duration head predicts the number of tokens it covers:

$$p_{\psi}(L_k | z_k) = \text{softmax}(\text{MLP}_{\text{dur}}(W_{\text{in}} z_k)). \quad (3.16)$$

Given the span lengths, the latents are expanded into a token-aligned condition $H^Z = (h_1^Z, \dots, h_T^Z)$. If $g(t) = k$ indicates that token t belongs to span k , then

$$h_t^Z = \text{Expand}(z_{g(t)}, j_t, L_{g(t)}), \quad (3.17)$$

where j_t is the relative position of token t within its span. The block-causal MDM decoder D_{ψ} then predicts the target tokens in token space:

$$p_{\psi}(y_t | \mathcal{K}_t, H_t^Z) = \text{softmax}(\text{Head}(D_{\psi}(\mathcal{K}_t), H_t^Z)), \quad (3.18)$$

where $\mathcal{K}_t$ contains the historical clean blocks and the current masked block; attention is causal across blocks and bidirectional denoising is allowed within the current block, which cannot read its own clean copy. The attention mask and training objective are detailed in Appendix G.3.

Latent generator. The latent sequence is partitioned into contiguous blocks of fixed size B ,

$$\mathcal{B}_j = \{(j-1)B + 1, \dots, \min(jB, K)\}, \quad J = \left\lceil \frac{K}{B} \right\rceil. \quad (3.19)$$

The generator G_θ is a block-causal DiT: causal across blocks and bidirectional within the current block. It learns $p_\theta(Z | C)$ in continuous latent space via Flow Matching. High-level generation therefore iterates along a latent axis of length $K \approx \rho T$, while discrete details are recovered by D_ψ in token space.

3.2.2 Training Procedure and Objectives

Stage 1: semantic compression and reconstruction-generation co-adaptation. Stage 1 jointly optimizes E_ϕ , the latent-to-token interface, and D_ψ . To cover the spectrum in Eq. (3.9) from strong-condition reconstruction to weak-condition generation, we sample one of three mutually exclusive condition-reliability states for each latent block. Let $A \in \{\mathbf{cond}, \mathbf{noise}, \mathbf{drop}\}$ and define

$$\tilde{Z}_A = \begin{cases} Z, & A = \mathbf{cond}, \\ (1 - u^c)Z + u^c \varepsilon, & A = \mathbf{noise}, \\ \emptyset, & A = \mathbf{drop}, \end{cases} \quad \varepsilon \sim \mathcal{N}(0, I), \quad (3.20)$$

where $\emptyset$ removes only the latent condition of the current block while retaining the same discrete token context. Let $\mathcal{L}_{\text{tok}}^A$ denote the MDM prediction objective under state A . The three states form

$$\mathcal{L}_{\text{bal}} = \lambda_c \mathcal{L}_{\text{tok}}^{\mathbf{cond}} + \lambda_n \mathcal{L}_{\text{tok}}^{\mathbf{noise}} + \lambda_d \mathcal{L}_{\text{tok}}^{\mathbf{drop}}, \quad (3.21)$$

where $\mathcal{L}_{\text{tok}}^A$ is the masked denoising loss under the corresponding state, defined in Appendix G.3.

To increase the generative information in Eq. (3.8), we additionally introduce

$$\mathcal{L}_{\text{gen}} = \mathcal{L}_{\text{encmask}}, \quad (3.22)$$

which masks part of the encoder input under a fixed span layout and directly supervises the encoder through the corresponding token prediction. Thus,

$$\mathcal{L}_{\text{DGD}} = \mathcal{L}_{\text{bal}} + \lambda_{\text{gen}} \mathcal{L}_{\text{gen}}. \quad (3.23)$$

The KL regularization of the diagonal Gaussian posterior is

$$\mathcal{L}_{\text{KL}} = \frac{K_{\text{ref}} d_{\text{ref}}}{T_{\text{ref}}} \frac{1}{K d_z} \sum_{k=1}^K \text{KL}(q_\phi(z_k | C, Y) \parallel \mathcal{N}(0, I)), \quad (3.24)$$

and the duration predictor is supervised with ground-truth span lengths, giving $\mathcal{L}_{\text{dur}}$. The unified Stage 1 objective is

$$\mathcal{L}^{(1)} = \mathcal{L}_{\text{DGD}} + \beta_{\text{KL}} \mathcal{L}_{\text{KL}} + \lambda_{\text{dur}} \mathcal{L}_{\text{dur}}. \quad (3.25)$$

Its key gradient relation is

$$\nabla_\phi \mathcal{L}_{\text{DGD}} = \frac{\partial \mathcal{L}_{\text{DGD}}}{\partial Z} \frac{\partial Z}{\partial \phi} \neq 0, \quad (3.26)$$

i.e., discrete token prediction directly shapes the encoder latents, so that E_ϕ and D_ψ jointly determine the information allocation in Eq. (3.10). This pathway remains active in Stage 2.

Stage 2: detached latent diffusion co-training. Stage 2 retains Eq. (3.25) and adds only detached latent-prior fitting. Let $\bar{Z} = \text{sg}(Z)$; for the j -th latent block, we sample

$$u_j = \sigma(\xi_j), \quad \xi_j \sim \mathcal{N}(\mu_u, \sigma_u^2), \quad (3.27)$$

and all latents within the block share the same u_j . Under the linear Flow Matching path,

$$z_u = (1 - u)\bar{z} + u\epsilon, \quad v^* = \epsilon - \bar{z}, \quad \epsilon \sim \mathcal{N}(0, I), \quad (3.28)$$

which gives

$$\mathcal{L}_{\text{FM}} = \mathbb{E} \left[\|G_\theta(z_u, u; \bar{Z}_{<j}) - v^*\|_2^2 \right]. \quad (3.29)$$

The total Stage 2 objective is therefore

$$\mathcal{L}^{(2)} = \mathcal{L}^{(1)} + \lambda_{\text{FM}} \mathcal{L}_{\text{FM}}, \quad (3.30)$$

and hence

$$\nabla_{\phi} \mathcal{L}_{\text{FM}} = 0, \quad \nabla_{\phi} \mathcal{L}_{\text{DGD}} \neq 0, \quad (3.31)$$

i.e., prior fitting is decoupled from representation learning, while the continuous representation and the discrete realization remain co-adapted end to end. The complete gradient routing is given in Appendix G.5.

3.2.3 Inference Procedure

Propose–decode–re-encode–commit. Given a prompt C , the encoder takes the posterior mode to obtain K_{p} prompt latents. Writing

$$K_{\text{p}} = B \left\lfloor \frac{K_{\text{p}}}{B} \right\rfloor + r, \quad 0 \leq r < B, \quad (3.32)$$

only the complete block-aligned prefix is persistently committed; the trailing r latents and their tokens serve as the clean prefix of the first generated block, and the remaining $B - r$ latent slots are initialized from Gaussian noise.

The j -th generated block follows a unified transaction

$$Z_j^{\text{prop}} \xrightarrow{D_{\psi}} Y_j \xrightarrow{E_{\phi}(\cdot; \hat{\mathcal{S}}_j)} Z_j^{\text{corr}} \xrightarrow{\text{commit}} \mathcal{K}_{j+1}, \quad (3.33)$$

where $\hat{\mathcal{S}}_j$ is the span layout already determined by the duration predictor; feedback does not rerun the dynamic segmenter. The proposal serves only the current token block, and the persistent latent history always stores Z_j^{corr} .

This design first preserves train–inference consistency. During training, the clean history is obtained by encoding ground-truth tokens; at inference,

$$Z_{\leq j}^{\text{hist}} = E_{\phi}(Y_{\leq j}; \hat{\mathcal{S}}_{\leq j}), \quad (3.34)$$

so the latent history read by the next block is still produced by the same encoder path as in training.

It also keeps the continuous latent context aligned with the discrete token context:

$$\mathcal{K}_{\leq j}^Z \equiv E_{\phi}(\mathcal{K}_{\leq j}^Y; \hat{\mathcal{S}}_{\leq j}). \quad (3.35)$$

If Z_j^{prop} were committed directly, the subsequent DiT history and the token history actually generated by the decoder could stem from different semantic states, causing cumulative context bias; fixed-span feedback projects both back onto the same committed history. The cache protocol is detailed in Appendix G.6.

Two-level CFG. The latent DiT modulates the historical latent condition in velocity space:

$$\hat{v}_{\gamma} = v_{\text{u}} + \gamma_{\text{gen}}(v_{\text{c}} - v_{\text{u}}), \quad (3.36)$$

where v_{c} and v_{u} are the velocity outputs with and without the historical latent condition, respectively, and γ_{gen} is the guidance scale. Its local probabilistic interpretation is given in Appendix F.

The MDM decoder uses `masked_only` conditioning, so the historical token KV contains no latent condition, and applies prefix–latent two-axis CFG in logit space. Fixing the current masked block and the span layout, let ℓ_{ab} denote the logits obtained by keeping (1) or masking (0) the historical token prefix and the current latent condition, where $a, b \in \{0, 1\}$ correspond to the two conditions, respectively. With two-axis scales $\gamma_{\text{p}}, \gamma_{\text{z}}$, we define

$$\begin{aligned} \ell_{\gamma_{\text{p}}, \gamma_{\text{z}}} &= \gamma_{\text{p}} \gamma_{\text{z}} \ell_{11} + \gamma_{\text{p}} (1 - \gamma_{\text{z}}) \ell_{10} \\ &\quad + (1 - \gamma_{\text{p}}) \gamma_{\text{z}} \ell_{01} + (1 - \gamma_{\text{p}}) (1 - \gamma_{\text{z}}) \ell_{00}. \end{aligned} \quad (3.37)$$

Setting $\gamma_p = 1$ modulates only the latent, setting $\gamma_z = 1$ modulates only the prefix, and setting both to 1 recovers the fully conditioned branch. This rule changes neither the training objective nor the feedback protocol. The four-branch definition and cache implementation are given in Appendices F.3 and G.6.1.

Let S_{gen} be the number of DiT denoising steps. On the uniform time grid $u_i = 1 - i/S_{\text{gen}}$, the explicit Euler update is

$$Z_j^{(i+1)} = Z_j^{(i)} - \frac{1}{S_{\text{gen}}} \hat{v}_\gamma(Z_j^{(i)}, u_i), \quad i = 0, \dots, S_{\text{gen}} - 1. \quad (3.38)$$

Given the proposal, D_ψ completes the current token block with a few iterative demasking steps, followed by the feedback correction in Eq. (3.33).

3.3 Computational Cost

Let D_s be the training token budget of Stage s , $c_{\text{rep}}^{(s)}$ the average per-token cost of the encoder-decoder path in that stage, and $c_G^{(2)}$ the average per-token cost of the added DiT, both including the corresponding attention and output projections. Then

$$\mathcal{C}_{\text{train}} \approx D_1 c_{\text{rep}}^{(1)} + D_2 (c_{\text{rep}}^{(2)} + c_G^{(2)}). \quad (3.39)$$

Under the same prior architecture and a comparable layout, $K = \rho T$ reduces the dominant linear-layer and attention-pair terms to $\mathcal{O}(\rho)$ and $\mathcal{O}(\rho^2)$ of their token-axis counterparts, respectively, but the representation path persists in both stages.

Inference cost consists of DiT proposal/commit, discrete decoding (including condition preparation), and fixed-span feedback, denoted by $\mathcal{C}_G^{\text{infer}}$, $\mathcal{C}_D^{\text{infer}}$, and $\mathcal{C}_{\text{fb}}$, respectively. Taking as reference the AR cost $\mathcal{C}_{\text{AR}}^{\text{infer}}$ under the same prompt/output budget and accounting convention,

$$R_{\text{infer}} = \frac{\mathcal{C}_G^{\text{infer}} + \mathcal{C}_D^{\text{infer}} + \mathcal{C}_{\text{fb}}}{\mathcal{C}_{\text{AR}}^{\text{infer}}}. \quad (3.40)$$

Both sides exclude the one-time prefill but retain the cost of subsequently reading the prompt KV.

Let ρ_{gen} be the realized latent/token ratio of the generated text, and let H_D and H_F be the numbers of per-block call rounds for condition preparation/decoding/clean-token commit and for feedback/corrected-cache commit, respectively. Ignoring boundary effects and assuming ideally parallel CFG branches, the number of serial call rounds $\mathcal{H}$ satisfies

$$\frac{\mathcal{H}_{\text{Cola2}}}{\mathcal{H}_{\text{AR}}} \approx \frac{\rho_{\text{gen}}}{B} (S_{\text{gen}} + H_D + H_F). \quad (3.41)$$

This is a forward-round proxy rather than a latency guarantee. The end-to-end cost also depends on the actual number of CFG branches, the token decoder, and feedback; per-term counts, the AR comparison, and the KV analysis are given in Appendix H.

4 Experiments

Following the training pipeline in Section 3.2, our experiments address three questions: (1) how to build a compressed latent space that is semantically rich and easy to model; (2) how to train the latent generator; and (3) how the decoder should allocate its capacity between conditional reconstruction and autonomous generation.

4.1 Experimental Setup

Datasets. For the component analyses and controlled scaling experiments, we train on the open-source OLMo 2 pretraining corpus [35]. The component analyses are conducted mainly on LAMBADA and examine the construction of the semantic latent space, the learning of the latent generator, and the reconstruction-generation balance of the decoder. The controlled scaling experiments are conducted on five downstream tasks, RACE [21], OBQA [31], Story Cloze [32], LAMBADA [37], and SQuAD [40], and compare how different

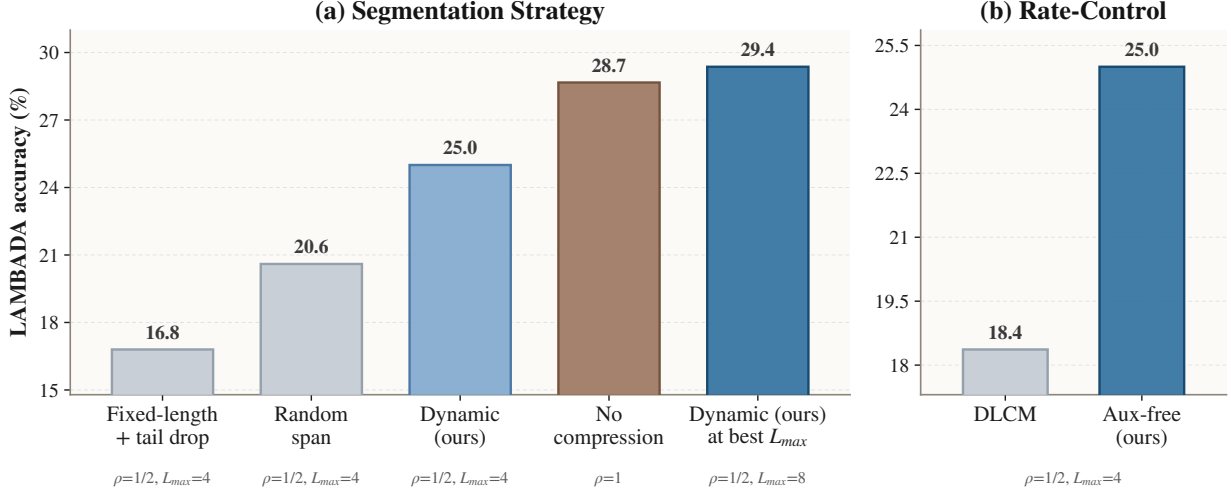


Figure 2 Segmentation and rate control. Dynamic segmentation outperforms fixed-length and random segmentation; auxiliary-free negative-feedback control outperforms DLCM [39].

generation paradigms improve as training scales. The post-pretraining comparison additionally evaluates HellaSwag [49], NarrativeQA [20], ARC-Easy [9], TruthfulQA [25], and MMLU [14].

Baselines. Component studies compare *Cola2* variants and DLCM [39] for rate control. Controlled scaling compares AR [6], LLaDA [34], Cola DLM [13], and the uncompressed Cola-MDM ($\rho = 1$) under matched model size, data, and evaluation; Cola-MDM shares the *Cola2*-MDM decoder to isolate the effect of compression. Post-pretraining comparisons use Llama-3-8B [11], Qwen2.5-7B [17], Qwen3-8B-Base [45], and LLaDA-8B-Base [34].

Metrics. Answers are extracted from model outputs according to each task format, and accuracy (%) is computed by exact match. All tasks except LAMBADA use few-shot prompting. The internal ablations report LAMBADA accuracy by default, and the controlled scaling experiments report the accuracy on all five tasks and their average.

Settings. In the component analyses and controlled scaling experiments, we use the OLMo 2 (dolma2) tokenizer and the AdamW optimizer with $\beta = (0.9, 0.95)$, weight decay 0.01, and a maximum sequence length of 512. Stage 1 jointly trains the encoder and decoder; Stage 2 trains the DiT while continuing to update the VAE, and blocks the Flow Matching gradient from reaching the encoder. The learning rates of the Stage 1 VAE and the Stage 2 DiT both start at 10^{-6} , increase linearly over 5,000 warmup steps to 10^{-4} and 1.5×10^{-4} , respectively, and then follow cosine decay to 10^{-6} at step 50,000 and 10^{-5} at step 1,000,000, respectively. In Stage 2, the VAE learning rate continues from Stage 1 with a 5,000-step transition, and the encoder learning rate is $0.1 \times$ that of the decoder. Unless otherwise stated, each latent covers 2 tokens on average, the latent dimension is 64, and the maximum span length is 8. DiT noise times follow a logit-normal distribution ($\text{loc} = 2.0$, $\text{scale} = 1.0$); the decoder samples clean, noisy, and dropped latent conditions per block with probabilities 0.2, 0.4, and 0.4, respectively. Inference uses 8 DiT denoising steps and 4 MDM demasking steps by default, and the controlled scaling experiments train on about 210B tokens.

4.2 Building the Semantic Latent Space

4.2.1 Dynamic Segmentation and Rate Control

Obs. 1 Dynamic segmentation is essential. With $\rho = 1/2$ and $L_{max} = 4$, dynamic segmentation (25.0%) outperforms random spans and fixed-length segmentation by 4.4 and 8.2 points (Figure 2); relaxing L_{max} to 8 reaches 29.4%, exceeding the uncompressed model (28.7%), which shows that content-based compression is not only lossless but also beneficial.

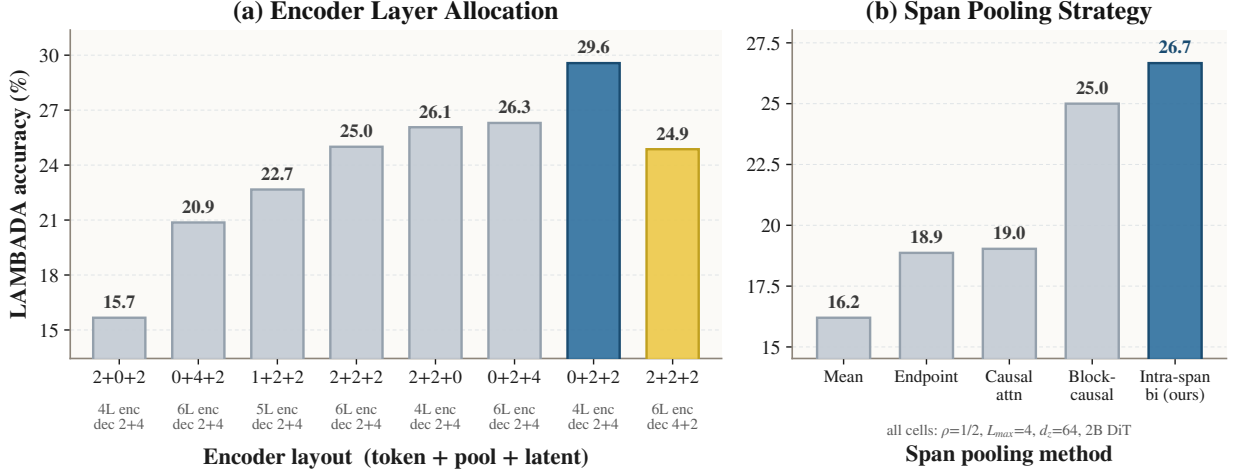


Figure 3 VAE architecture. Left: the allocation of encoder layers matters more than depth, with 0+2+2 performing best. Right: intra-span bidirectional attention is the best pooling operator.

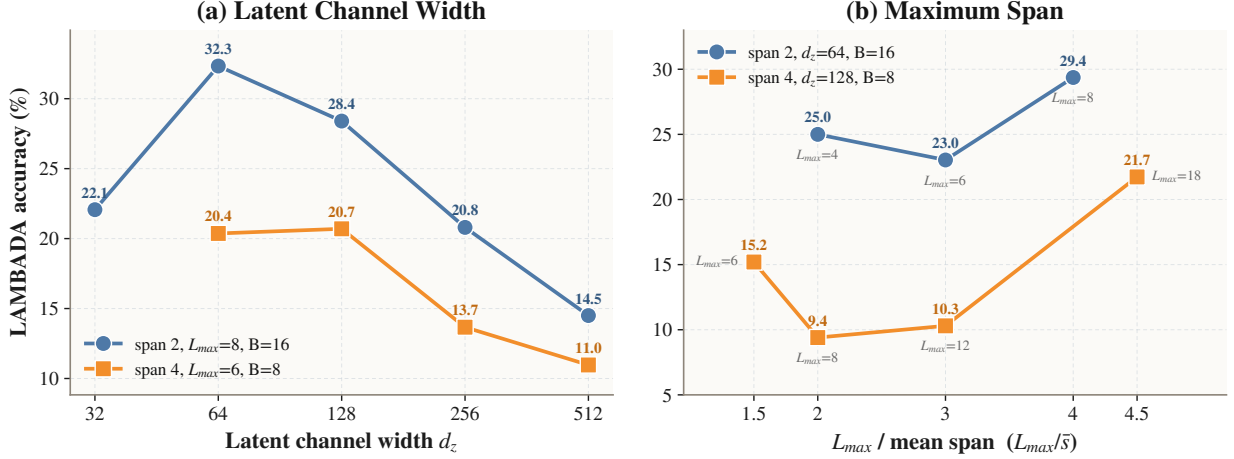


Figure 4 Latent width and maximum span length. Left: d_z has an interior optimum. Right: the effect of L_{max} is non-monotonic.

Obs. 2 Feedback rate control beats an auxiliary rate loss. Under the same setting, the former reaches 25.0% whereas DLCM [39] reaches only 18.4%: directly regulating the hard compression rate avoids the competition between the rate loss and representation learning.

4.2.2 VAE Architecture

Obs. 3 Encoder layout matters more than size. The four-layer 0+2+2 encoder achieves the best result of 29.6%, outperforming all six-layer variants (20.9–26.3%); removing the pooling layers (2+0+2) yields only 15.7%, whereas reallocating decoder layers has almost no effect (25.0% vs. 24.9%; Figure 3, left). An oversized encoder makes the latent distribution more complex and harder for the DiT to model. Intra-span bidirectional attention performs best (26.7%), followed by block-causal attention (25.0%), while fully causal attention, endpoint pooling, and mean pooling all stay at or below 19.0% (Figure 3, right).

4.2.3 Compression and Latent Capacity

Obs. 4 Wider latents are not always better. With mean span length 2, accuracy peaks at 32.3% at $d_z = 64$ and drops to 14.5% at $d_z = 512$ (Figure 4, left). A larger d_z adds capacity, but also increases the complexity

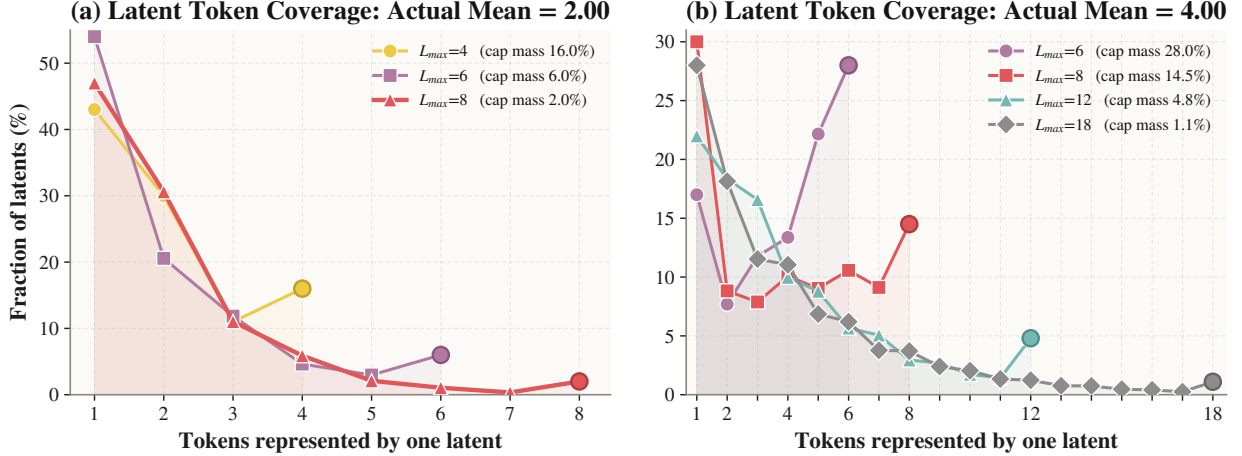


Figure 5 Distribution of the number of tokens covered by each latent. Increasing $L_{\max}$ reduces the number of forcibly truncated spans.

Table 1 Best configuration at each compression level. Accuracy decreases as compression becomes stronger.

Mean span length	Latent width d_z	$L_{\max}$	LAMBADA (%)
2	64	8	32.3
3	128	8	26.9
4	256	18	25.2

of the distribution that the DiT must fit.

Obs. 5 Compression rate, d_z , and $L_{\max}$ must be tuned jointly. Even with the best configuration at each compression level, accuracy still falls from 32.3% (span length 2) to 25.2% (span length 4), as shown in Table 1. The effect of $L_{\max}$ (Figure 4, right) is governed by the truncation ratio (Figure 5): when truncation is frequent, a smaller $L_{\max}$ keeps each latent simpler; once the cap mass drops to about 1% (span length 4, $L_{\max} = 18$), preserving complete semantic spans dominates and accuracy rises to 21.7%.

4.2.4 Generative Supervision of the Encoder

Lemma 4.1 (Generative Supervision Promotes Predictive Representations). *When the supervised tokens are hidden from the encoder, the generative objective cannot be satisfied by copying the current targets. When the encoder is trainable, reducing this objective encourages the latent to retain information predictive of missing or future tokens. If the encoder is frozen, the objective only improves the decoder’s use of information already present in the latent. See Appendix B for the proof.*

Obs. 6 Generative supervision is key; its form is secondary. Pure reconstruction reaches only 15.7%, and adding the Span-AR or BERT-style objective raises accuracy to 25.2% and 26.5%, respectively (Figure 6). With a frozen encoder, the BERT-style objective yields a 2.2-point gain (28.9% $\rightarrow$ 31.1%), and unfreezing the encoder reaches 35.2%. Consistent with Lemma 4.1, the generative objective improves both the completion ability of the decoder and the latent itself.

Obs. 7 Encoder and decoder should co-evolve. With the MDM decoder, a frozen encoder yields 47.6%, end-to-end co-evolution raises accuracy to 53.5%, and further adding the generative loss reaches 58.2% (Figure 6, right). What the latent should retain must be determined jointly with what the decoder can complete.

4.3 Learning the Latent Generator

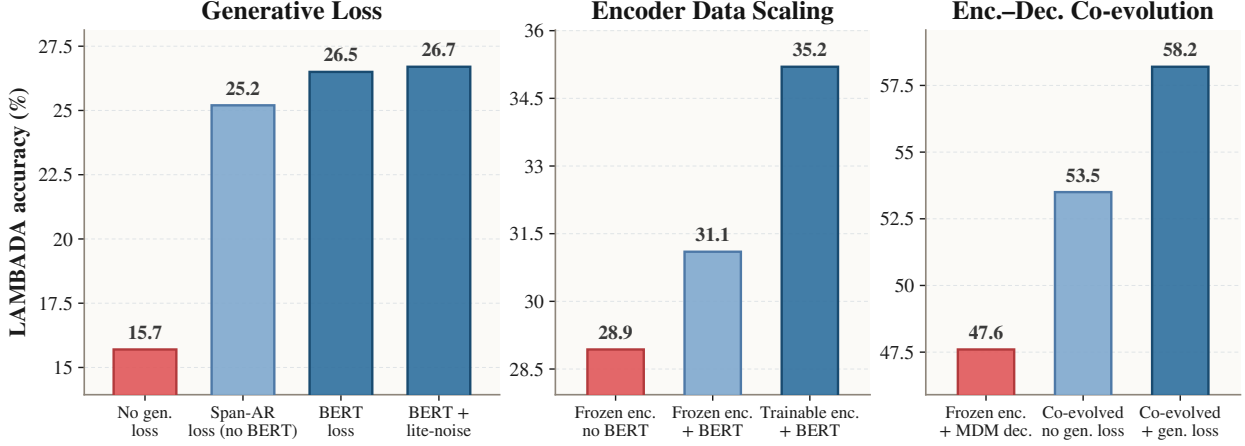


Figure 6 Generative supervision. Left: generative objectives substantially outperform pure reconstruction. Middle: the gain is larger when the encoder is trainable. Right: co-evolving the encoder and decoder with the generative loss performs best.

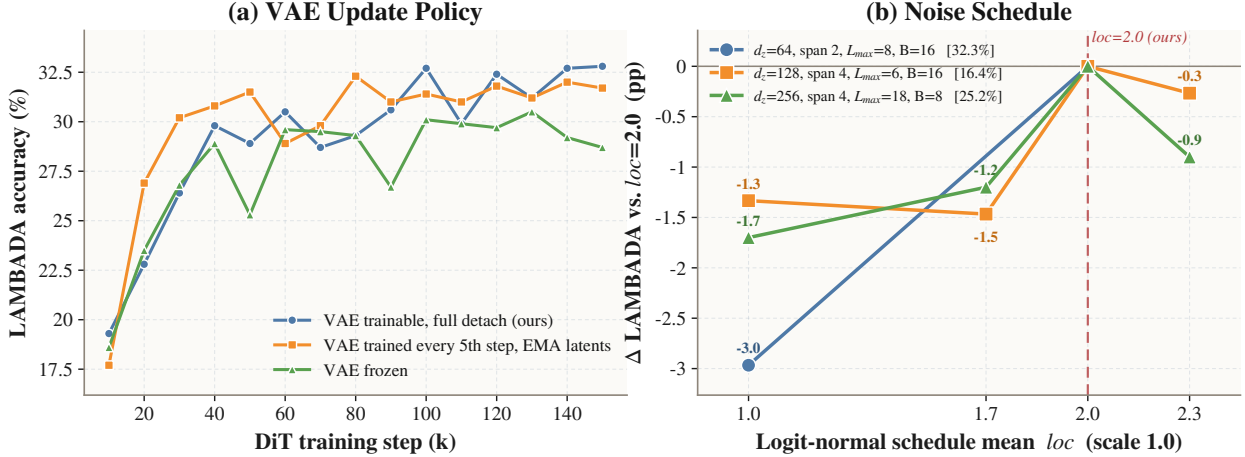


Figure 7 VAE update policy and noise schedule. Left: continuing to train the VAE while detaching the DiT gradient performs best. Right: differences relative to $loc = 2.0$; all three configurations are optimal at 2.0.

4.3.1 VAE Update Policy and Gradient Detachment

Lemma 4.2 (Prior Fitting and Representation Learning Should Be Decoupled). *The DiT Flow Matching objective only requires the current latent to be easier for the prior to fit; its gradient is not guaranteed to align with the token-side representation objective. Applying stop-gradient to the DiT path ensures that prior fitting does not directly rewrite the encoder, while preserving end-to-end encoder updates from the decoder token loss. See Appendix C for the proof.*

Obs. 8 Keep training the VAE, but detach the DiT gradient. A trainable VAE reaches 32.8%, whereas a frozen VAE reaches only 28.7% (Figure 7, left). Allowing the DiT gradient to flow into the encoder yields a 2.2-point lead early on, but this is overtaken after 80k steps and becomes a 1.0-point deficit at 140k steps (Table 2). This supports Lemma 4.2: the DiT gradient rewrites the latent to be easier to fit rather than more useful for text modeling, and detaching the gradient blocks this latent hacking.

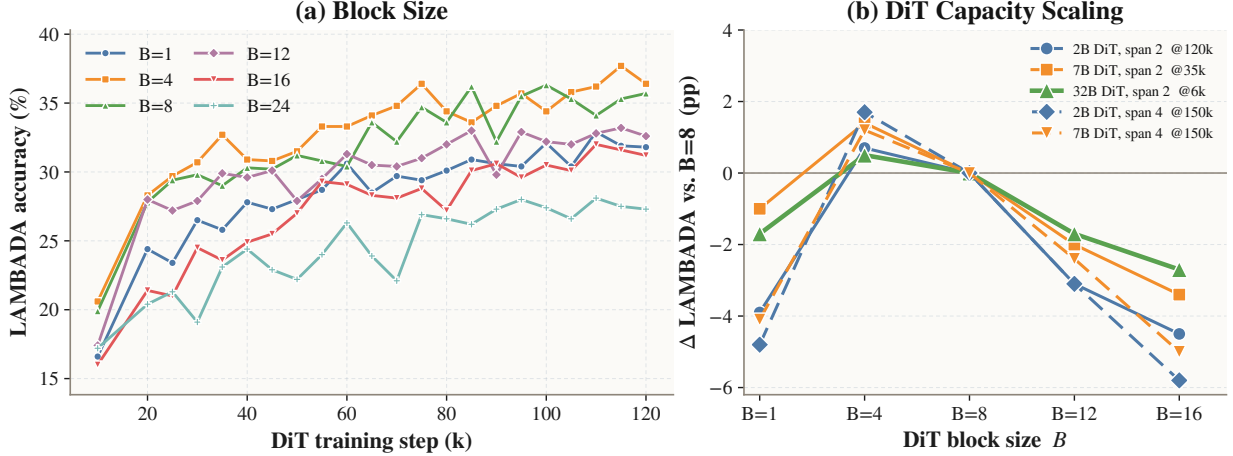


Figure 8 DiT block size and capacity. Left: moderate block sizes are optimal. Right: larger DiTs yield flatter curves.

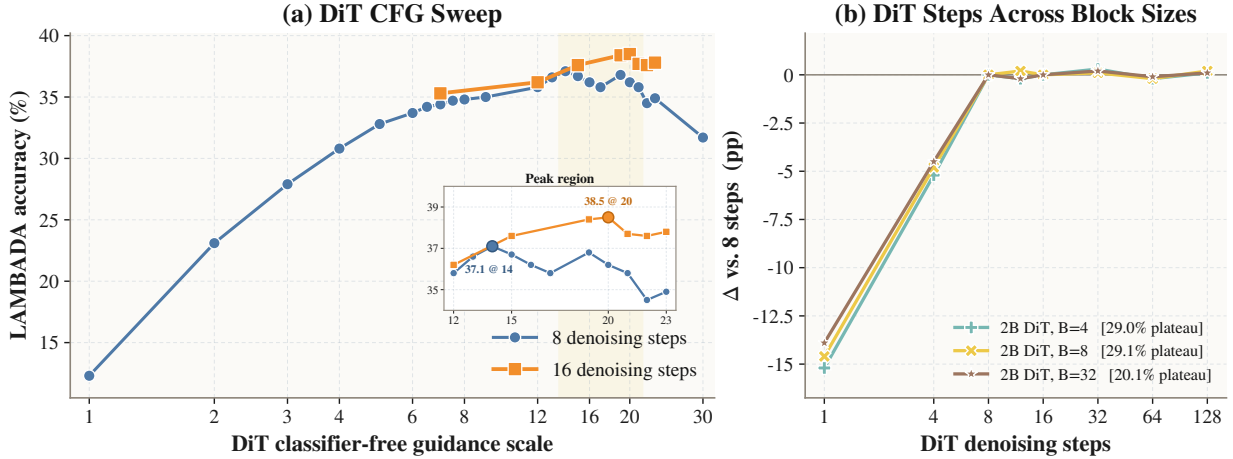


Figure 9 DiT inference. Left: generator CFG sweep. Right: the DiT saturates at 8 steps.

4.3.2 Noise-Schedule Calibration

Lemma 4.3 (Effective Noise Is Jointly Determined by the Latent and Diffusion). *At the same DiT timestep, the available latent signal is affected by both VAE posterior uncertainty and diffusion corruption. Consequently, changing the noise scale of the Stage 1 latent changes the effective signal-to-noise ratio induced by the same DiT schedule, so the noise schedule must be calibrated jointly with the latent distribution. See Appendix D for the proof.*

Obs. 9 The noise schedule must match the representation. All three configurations achieve their best results at $\text{loc} = 2.0$, and deviating from it costs 0.3–3.0 points, with the magnitude varying with d_z and the compression level (Figure 7, right). By Lemma 4.3, the schedule should be recalibrated when the representation changes.

4.3.3 Block Size and Generator Capacity

Obs. 10 Moderate blocks are best, and larger DiTs tolerate larger blocks. For the 2B DiT, $B = 4$ is optimal (36.4%), whereas $B = 1$ and $B = 24$ reach only 31.8% and 27.3%, respectively (Figure 8, left). In the right panel of Figure 8, the slope of the decline for $B > 8$ becomes monotonically flatter with DiT size: the loss from $B = 8 \rightarrow 16$ decreases from 4.5 points for 2B to 3.4 points for 7B and 2.7 points for 32B, and the same trend holds at mean span length 4. A larger DiT can jointly model more latents within a single block, which enlarges the range of block sizes it can use effectively.

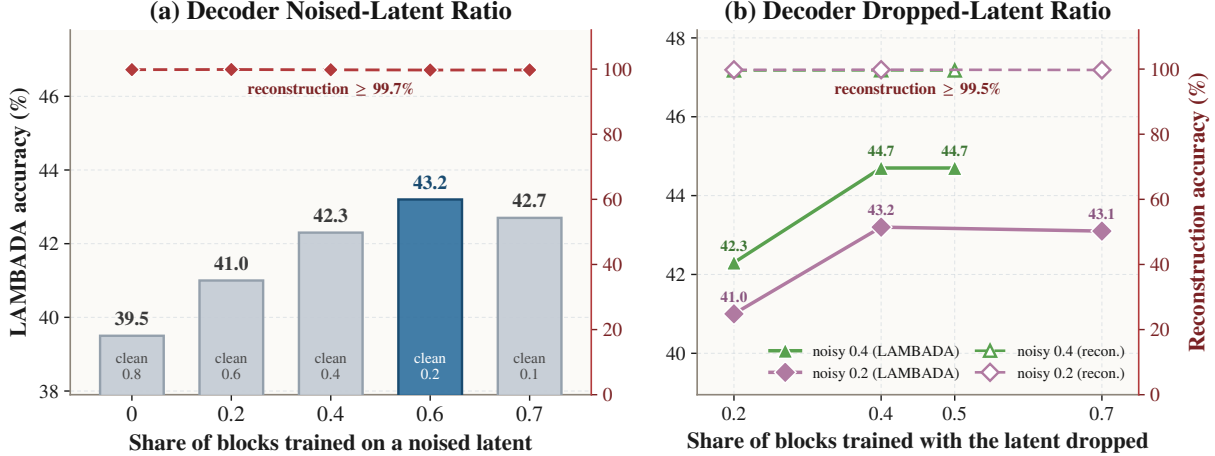


Figure 10 Decoder data partition. Left: varying the noisy-to-clean ratio. Right: varying the drop ratio. Reconstruction accuracy (right axis) stays above 99.5% throughout; 0.4/0.4/0.2 gives the best balance.

Table 2 Whether the DiT gradient flows into the encoder. Gradient injection leads early, but is overtaken by full detachment in the middle and late stages of training.

Gradient policy	20k steps	80k steps	140k steps
Full detachment (ours)	21.4	30.4	33.1
DiT velocity loss injected every 10 steps	23.6	29.9	32.1
Difference (ours – injection)	−2.2	+0.5	+1.0

4.3.4 Generator Guidance and Inference Steps

Obs. 11 The generator needs strong CFG and about 8 steps. At 8 steps, accuracy rises from 12.2% at $\gamma_{\text{gen}} = 1$ to 37.1% at $\gamma_{\text{gen}} = 14$; at 16 steps, the peak moves to $\gamma_{\text{gen}} = 20$ (38.5%) (Figure 9, left). Using 1 and 4 steps loses about 14 and 5 points, respectively, whereas results between 8 and 128 steps differ by at most 0.3 points (Figure 9, right).

4.4 Balancing the Reconstruction--Generation Decoder

Lemma 4.4 (Latent-Condition Strength Determines Residual Generation). *Given the same discrete context, a more reliable latent condition provides more information and leaves less residual uncertainty for the decoder to generate on its own; when the latent is corrupted or removed, the residual uncertainty increases accordingly. Thus, mixed **cond/noise/drop** training covers the continuum from conditional reconstruction to autonomous generation. See Appendix E for the proof.*

4.4.1 Decoder Type

Obs. 12 Decoder factorization matters little. At every checkpoint, the gap between them is at most 1.4 points, and the leader alternates during training (Table 3): AR-MTP leads by 0.9 points at 50k steps, whereas MDM leads by 0.7 and 1.4 points at 100k and 150k steps. This indicates that, given a shared latent-token interface, the specific decoder factorization is not the decisive factor for performance, and the hierarchical framework applies to both; subsequent experiments use the MDM decoder.

4.4.2 Decoder Data Partition

Obs. 13 Train mostly on noisy and dropped conditions, but keep clean ones. Reconstruction accuracy stays above 99.5% in all configurations, even when only 10% of blocks are clean (Figure 10), so most of the data can

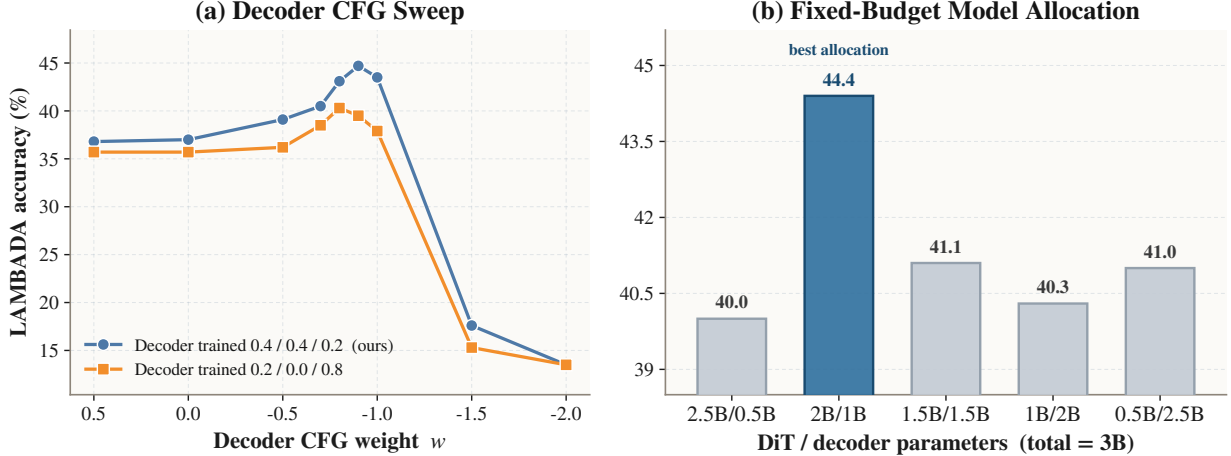


Figure 11 Decoder guidance and parameter allocation. Left: the balanced decoder is optimal at $w_{\text{dec}} = -0.9$. Right: with 3B total parameters, a 2B DiT with a 1B decoder is optimal.

Table 3 Decoder factorization. The two decoders perform similarly.

Decoder	50k steps	100k steps	150k steps
AR-MTP decoder	39.7	43.7	44.1
MDM decoder	38.8	44.4	45.5
Difference (MDM – AR-MTP)	−0.9	+0.7	+1.4

be devoted to generation under degraded conditions. With the drop ratio fixed at 0.2, shifting noisy/clean from 0.0/0.8 to 0.6/0.2 raises LAMBADA accuracy from 39.5% to 43.2%; raising the drop ratio from 0.2 to 0.4 adds another 2.2–2.4 points, while pushing further brings no gain (noisy 0.7: 42.7%; drop 0.7: 43.1%). The selected 0.4/0.4/0.2 (drop/noisy/clean) ties for the best result of 44.7% while retaining more clean data, confirming the reconstruction–generation continuum in Lemma 4.4.

4.4.3 Decoder Guidance

Obs. 14 Decoder CFG must match the training mixture. The balanced decoder reaches 44.7% at $w_{\text{dec}} = -0.9$ and collapses rapidly at more negative weights, whereas the clean-heavy decoder reaches at best 40.3% (Figure 11, left), indicating that inference-time guidance cannot compensate for an imbalanced training distribution.

4.4.4 Decoder Capacity

Obs. 15 Under a fixed budget, the DiT matters more. With the DiT fixed, increasing the decoder from 500M to 2B yields monotonic gains of 2.1–2.4 points (Figure 12); under a total budget of 3B, a 2B DiT with a 1B decoder (44.4%) outperforms the other allocations by 3.3–4.4 points, and imbalance in either direction degrades performance (Figure 11, right).

4.4.5 Demasking Steps and Latent Use

Obs. 16 Demasking saturates at about 8 steps. 8 DiT/8 demasking steps achieve the best result of 49.0% (Figure 13, left); fewer than 8 DiT steps cause a clear drop, while increasing demasking can partially compensate when the DiT uses too few steps. The 8/4 setting (48.2%) is only 0.8 points below the optimum and serves as the default low-cost setting.

Obs. 17 The decoder relies on aligned latents. Zeroing, resampling, and shuffling the latent reduce accuracy

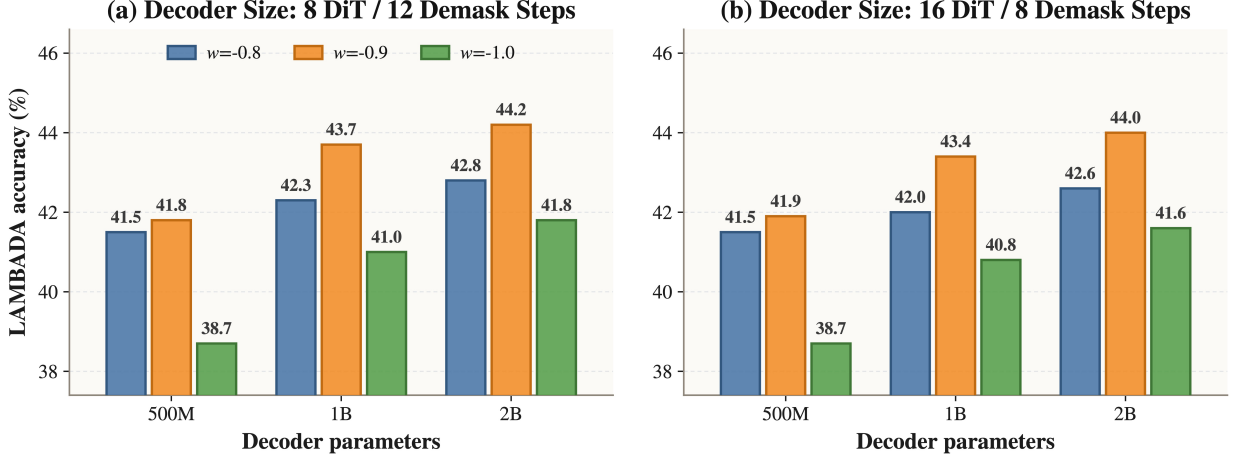


Figure 12 Scaling the decoder with a fixed DiT. Accuracy improves monotonically with decoder size.

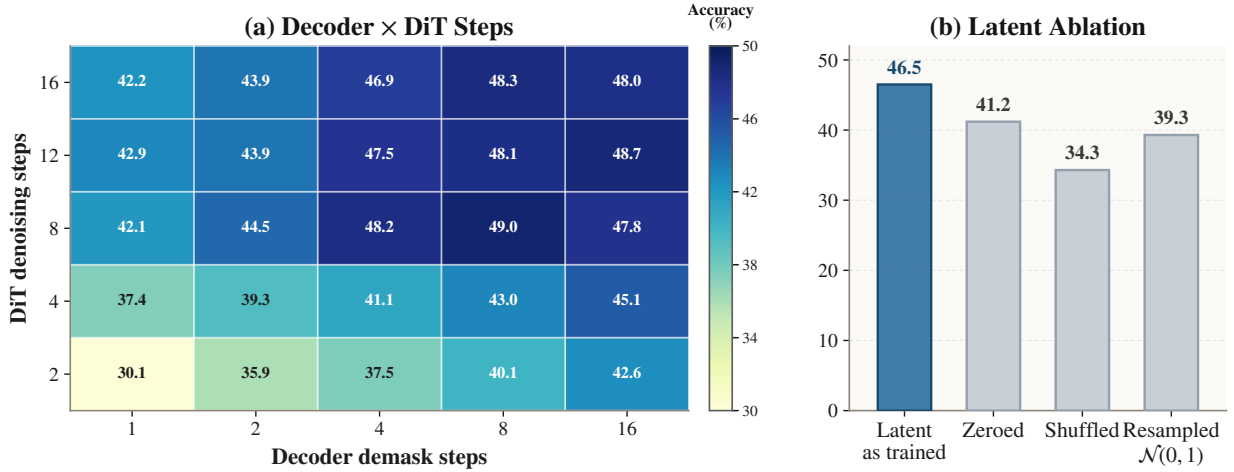


Figure 13 Decoder inference and latent ablation. Left: 8/8 steps are optimal. Right: corrupting the latent substantially degrades performance, with shuffling being the most harmful.

from 46.5% to 41.2%, 39.3%, and 34.3%, respectively (Figure 13, right). Zeroing corresponds to the **drop** state seen in training and therefore degrades more gracefully; shuffled latents are statistically realistic but semantically misaligned and cause the largest damage, showing that the decoder follows the latent content.

4.5 Scaling Experiments

4.5.1 Controlled Scaling Trends

Figure 14 extends training to about 210B tokens and compares the models on RACE, OBQA, Story Cloze, LAMBADA, and SQuAD. Cola-MDM ($\rho = 1$) and Cola2-MDM (with the compression VAE from Section 4.2) share the MDM decoder selected in Section 4.4, which separates the effect of the decoder design from that of dynamic compression.

Obs. 18 The MDM decoder yields the strongest scaling. Cola-MDM finally reaches a task average of about 34.5%, higher than AR (about 33%), and about 64% on LAMBADA, about 15 points above Cola DLM. The weakness of Cola DLM on generative tasks stems from its decoder rather than from the hierarchical framework.

Obs. 19 Compression preserves semantic tasks, with costs on exact-token tasks. Cola2-MDM matches AR on

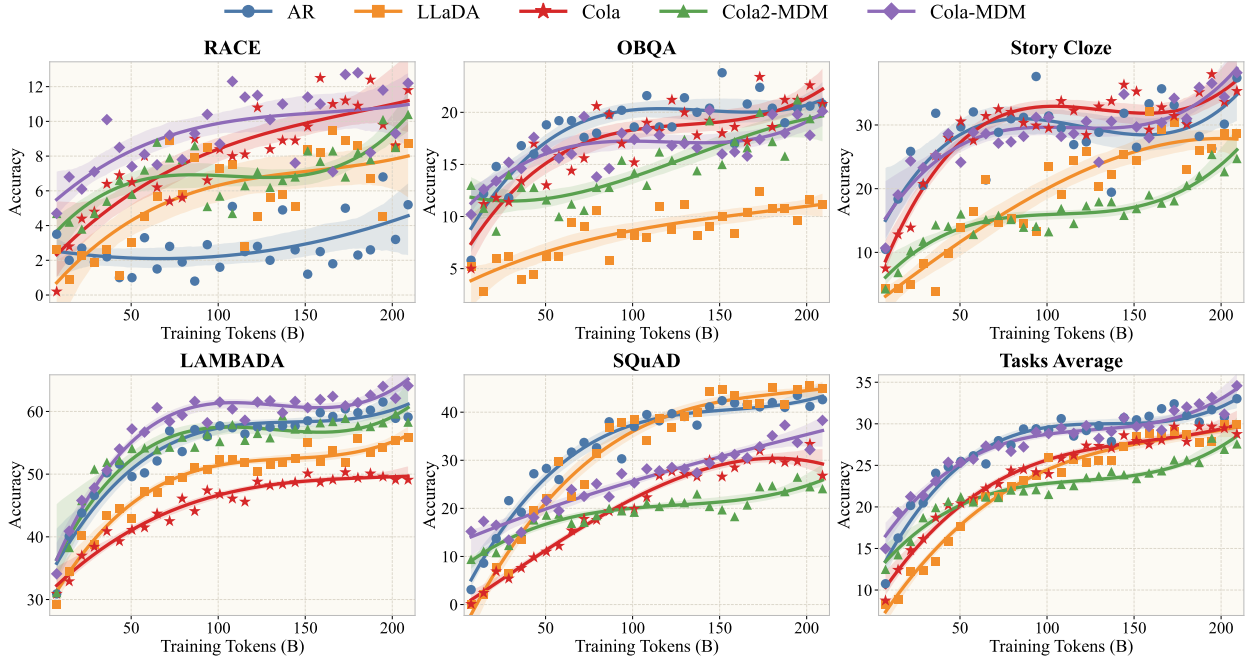


Figure 14 Scaling trends with training tokens. Cola-MDM achieves the strongest overall scaling; Cola2-MDM also exhibits strong and unsaturated scaling potential under compression.

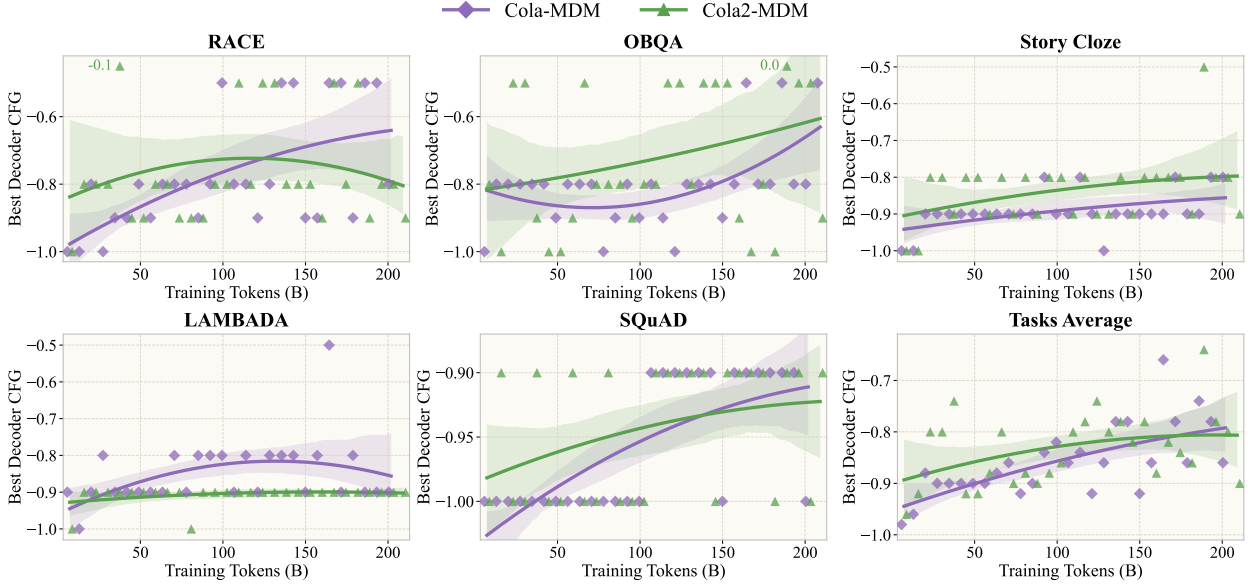


Figure 15 Optimal decoder CFG weight during training. For both models, the optimal weight remains within $[-1.0, -0.8]$, consistent with the -0.9 selected in Section 4.4, and drifts slowly toward 0 during training; the drift of Cola2-MDM is more gradual, consistent with its slower convergence.

LAMBADA (about 61%) and leads substantially on RACE (about 10.5% vs. 4.5%), with the gap concentrated on SQuAD and Story Cloze, which rely on verbatim reproduction.

Obs. 20 The compressed variant converges more slowly but keeps closing the gap. Since each latent summarizes more tokens, the latent DiT converges more slowly than the discrete generator. As training

Table 4 Post-pretraining evaluation. Accuracy (%) on eight downstream tasks, with tasks grouped by evaluation focus and models by generation paradigm. *Cola2* uses 346.03B source tokens; the models are not compute-matched.

Task	Autoregressive			Discrete diffusion	Continuous diffusion
	Llama-3-8B	Qwen2.5-7B	Qwen3-8B-Base	LLaDA-8B-Base	<i>Cola2</i>
<i>Pretraining tokens</i>	15T+	18T	~36T	2.3T	346B
<i>Reading comprehension</i>					
RACE	75.77	89.95	90.65	87.40	69.91
NarrativeQA	42.70	32.90	37.30	36.00	32.70
<i>Commonsense reasoning</i>					
HellaSwag	39.21	75.67	68.13	69.65	41.75
StoryCloze	79.49	95.09	97.44	95.94	80.42
<i>Language modeling</i>					
LAMBADA	75.70	71.82	71.20	67.90	63.00
<i>Knowledge-oriented QA</i>					
ARC-Easy	90.86	96.00	97.35	93.85	87.62
TruthfulQA	63.04	69.03	72.58	64.87	54.10
MMLU	63.19	70.16	72.31	61.27	51.32
8-task average	66.25	75.08	75.87	72.11	60.10

continues, the latent condition becomes increasingly useful: after about 150B tokens, the average gap to LLaDA shrinks from about 4 to 1.5 points without saturating. The mixed decoder setting $w_{\text{dec}} = -0.9$ (44.7%) also outperforms both the latent-free (43.5%) and fully conditioned (37.0%) settings, confirming that latent semantics and discrete generation are complementary.

4.5.2 Post-Pretraining Benchmark Comparison

The final checkpoint evaluated here uses a separate large-scale recipe from the controlled scaling study. It is trained on OLMo 2 pretraining data retokenized with the Qwen3-4B tokenizer, using length-4096 sequences. The decoder and the VAE encoder embedding are initialized from Qwen3-4B-Base. During decoder training, masked positions are represented by [MASK] inputs, while decoder self-attention remains causal. Stage I trains the dynamic-compression VAE on 52.43B source tokens. Stage II jointly updates the VAE and an approximately 4B-parameter latent DiT, with detached VAE latents, on an additional 125.83B source tokens. Stage III freezes the Stage-II EMA VAE and continues training only the DiT on 167.77B source tokens. The total is 346.03B source tokens. Architecture, optimization, objective, EMA, and state-transition details are provided in Appendix I.

Inference uses a low-noise-biased eight-step Euler DiT schedule, $t_i = T(1 - i/8)^2$ for $i = 0, \dots, 8$, with DiT CFG set to 10. Decoder CFG is -0.9 , each decoding call generates one latent span, dynamic duration and latent feedback are enabled, and decoding is greedy.

We compare this checkpoint with representative 7B–8B pretrained models across eight downstream tasks. Each *Cola2* task score reported below is the average of three sampled evaluations. Because the models’ training recipes, tokenizers, and token-counting conventions differ, this comparison contextualizes post-pretraining performance rather than isolating the effect of the generation paradigm.

Obs. 21 *Cola2* shows strong scaling potential under a limited training budget. With only 346.03B source tokens and before convergence, *Cola2* reaches an eight-task average of 60.10, only 6.15 points behind Llama-3-8B (66.25), which uses at least 15T tokens—over $43\times$ more data. *Cola2* also surpasses Llama-3-8B on HellaSwag (41.75 vs. 39.21) and StoryCloze (80.42 vs. 79.49), while reaching 87.62 on ARC-Easy. These results indicate substantial scaling headroom and successful transfer of the Qwen-initialized causal decoder’s text generation

capability. More broadly, continuous-latent hierarchical modeling can integrate effectively with established token-space generation paradigms, including autoregressive and masked-diffusion decoders.

4.6 Summary

The results support the design of *Cola2* in three respects. **Semantic compression:** dynamic segmentation and generative supervision keep the latents strongly semantic at high compression rates; the former slightly surpasses the uncompressed model with an appropriate $L_{\max}$, and the latter raises accuracy from 15.7% to 26.7%. **Reconstruction--generation balance:** mixed training dominated by noisy and dropped conditions enables the decoder both to faithfully use the latent (shuffling the latent costs 12.2 points) and to autonomously generate the residual information. **Continuous--discrete co-adaptation:** the information placed in the latent should be determined by the decoder; co-training the encoder and decoder yields a 5.9-point gain, and both continue to benefit from more training data, giving the hierarchical model scaling trends comparable to or better than AR and LLaDA.

5 Limitations & Future Prospects

Although our results provide initial evidence for the feasibility, competitiveness, and scaling potential of *Cola2*, we view the current system as a starting point rather than a finished endpoint. First, the model remains far from convergence, especially the latent DiT, while the present training scale is limited by available resources. Larger models, longer training, and broader data and compute budgets are therefore needed to establish its performance ceiling. Second, compression reduces the dominant linear and attention FLOPs of the latent prior in theory, but end-to-end generation speed also depends on the decoder, CFG branches, latent feedback, and hardware utilization. Infrastructure co-design—including specialized kernels, cache management, and parallel scheduling—is needed to translate theoretical savings into practical speedups. Third, *Cola2* already shows compatibility with autoregressive and masked-diffusion decoders, but their co-training objectives, inference trade-offs, and optimal division of labor remain underexplored. Finally, the continuous latent space may provide a shared interface between text and other continuous modalities such as vision, opening a path toward unified latent representations, cross-modal priors, and unified understanding and generation.

6 Conclusion

We presented *Cola2*, a hierarchical language modeling framework that unifies dynamic semantic compression with balanced discrete reconstruction and generation through continuous-discrete co-adaptation. Dynamic compression and generation-oriented supervision encourage compact, predictive latent representations, while the decoder realizes latent semantics and handles residual uncertainty from context. Token-prediction gradients drive encoder-decoder co-adaptation, whereas latent-prior fitting remains decoupled from representation learning. Systematic component analyses and controlled evaluations on five downstream tasks, scaled to about 210B training tokens, support these design principles and show competitive, unsaturated scaling. In a separate resource-constrained run, the final checkpoint trained on 346.03B source tokens achieves a 60.10 average over eight tasks and surpasses Llama-3-8B on HellaSwag and StoryCloze, although that baseline reports over $43\times$ more pretraining tokens. These results indicate successful transfer of the Qwen-initialized causal decoder’s generation capability and substantial scaling headroom, especially because the latent DiT remains far from convergence. More broadly, *Cola2* shows that continuous-latent hierarchical modeling can integrate effectively with established token-space generators, including autoregressive and masked-diffusion decoders, rather than depending on a single decoder paradigm.

References

- [1] Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pages 50726–50753, 2025.
- [2] 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.
- [3] Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R Costa-jussà, David Dale, et al. Large concept models: Language modeling in a sentence representation space. *arXiv preprint arXiv:2412.08821*, 2024.
- [4] Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.
- [5] Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.
- [6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [7] Jiaqi Cao, Chiyu Chen, Shuang Cheng, Xu Cheng, Beiya Dai, Yufan Feng, Kewen Ge, Ruijun Ge, Jiayi Huang, Yang Jiao, et al. Ncp-archpreview technical report: Moving towards latent space language models through next concept prediction. *arXiv preprint arXiv:2609.10715*, 2026.
- [8] Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. Langflow: Continuous diffusion rivals discrete in language modeling. *arXiv preprint arXiv:2604.11748*, 2026.
- [9] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- [10] Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022.
- [11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [12] Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.
- [13] Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, et al. Continuous latent diffusion language model. *arXiv preprint arXiv:2605.06548*, 2026.
- [14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- [15] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.
- [16] Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. Elf: Embedded language flows. *arXiv preprint arXiv:2605.10938*, 2026.
- [17] Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.
- [18] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. In *International Conference on Learning Representations*, volume 2026, pages 149273–149313, 2026.
- [19] Jiaxiu Jiang, Jingjing Ren, Wenbo Li, Bo Wang, Haoze Sun, Yijun Yang, Jianhui Liu, Yanbing Zhang, Shenghe Zheng, Yuan Zhang, Haoyang Huang, Nan Duan, and Wangmeng Zuo. TextLDM: Language modeling with continuous latent diffusion. *arXiv preprint arXiv:2605.07748*, 2026.

- [20] Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.
- [21] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 785–794, 2017.
- [22] Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026.
- [23] 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.
- [24] Jiajun Liang, Yucheng Liao, Yukang Cao, Jiazhe Wei, Ken Li, Wende Tan, Jiankun Zhang, ZY Cui, Jingkan Yang, Liucheng Guo, Shiqi Yang, B. Yang, Caifeng Shan, Ziwei Liu, and Chenyang Si. AURORA-LM: Autoencoding unified representation for continuous-latent diffusion language modeling. *arXiv preprint arXiv:2608.02602*, 2026.
- [25] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, 2022.
- [26] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.
- [27] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.
- [28] Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q Weinberger. Latent diffusion for language generation. *Advances in Neural Information Processing Systems*, 36:56998–57025, 2023.
- [29] Viacheslav Meshchaninov, Egor Chibulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry Vetrov. Cosmos: Compressed and smooth latent space for text diffusion modeling. *Advances in Neural Information Processing Systems*, 38:14271–14299, 2026.
- [30] Viacheslav Meshchaninov, Alexander Shabalin, Egor Chibulatov, Nikita Gushchin, Ilya Koziev, Alexander Korotin, and Dmitry Vetrov. How to train your latent diffusion language model jointly with the latent space. *arXiv preprint arXiv:2605.07933*, 2026.
- [31] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391, 2018.
- [32] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, 2016.
- [33] Piotr Nawrot, Szymon Tworowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571, 2022.
- [34] 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. *Advances in Neural Information Processing Systems*, 38: 50608–50646, 2026.
- [35] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.
- [36] Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, 2025.

- [37] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534, 2016.
- [38] 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.
- [39] Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, et al. Dynamic large concept models: Latent reasoning in an adaptive semantic space. *arXiv preprint arXiv:2512.24617*, 2025.
- [40] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392, 2016.
- [41] Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.
- [42] Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. *Proceedings of machine learning research*, 267:52584, 2025.
- [43] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [44] Roy Uziel, Omer Belhasin, Itay Levy, Akhiad Bercovich, Ran El-Yaniv, Ran Zilberstein, and Michael Elad. Crocodil: Continuous and robust conditioned diffusion for language. *arXiv preprint arXiv:2603.20210*, 2026.
- [45] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [46] Zhihan Yang, Wei Guo, Shuibai Zhang, Subham Sekhar Sahoo, Yongxin Chen, Arash Vahdat, Morteza Mardani, and John Thickstun. Continuous diffusion scales competitively with discrete diffusion for language. *arXiv preprint arXiv:2605.18530*, 2026.
- [47] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025. URL <https://arxiv.org/abs/2508.15487>.
- [48] Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, 36:78808–78823, 2023.
- [49] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.
- [50] Yizhe Zhang, Jiatao Gu, Zhuofeng Wu, Shuangfei Zhai, Joshua Susskind, and Navdeep Jaitly. Planner: Generating diversified paragraph via latent language diffusion model. *Advances in Neural Information Processing Systems*, 36: 80178–80190, 2023.
- [51] Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. Continuously augmented discrete diffusion model for categorical generative modeling. In *International Conference on Learning Representations*, volume 2026, pages 34212–34242, 2026.
- [52] Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *arXiv preprint arXiv:2510.03206*, 2025.

Appendix

A Detailed Derivations of the Probabilistic Model and Information Decomposition

A.1 Conditional ELBO

During training, the encoder $q_\phi(Z | C, Y)$ serves as the approximate posterior, where ϕ denotes the encoder parameters. By Jensen’s inequality, the conditional log-likelihood in Eq. (3.1) satisfies

$$\log p(Y | C) \geq \mathbb{E}_{q_\phi(Z | C, Y)} \left[\log p_\psi(Y | C, Z) - \log \frac{q_\phi(Z | C, Y)}{p_\theta(Z | C)} \right]. \quad (\text{A.1})$$

Define the conditional aggregated posterior as

$$\bar{q}_\phi(Z | C) = \int q_\phi(Z | C, Y) p_{\text{data}}(Y | C) dY. \quad (\text{A.2})$$

Adding and subtracting $\log \bar{q}_\phi(Z | C)$ in the average KL term gives

$$\begin{aligned} & \mathbb{E} \text{KL}(q_\phi(Z | C, Y) \| p_\theta(Z | C)) \\ &= I(Y; Z | C) + \mathbb{E}_C \text{KL}(\bar{q}_\phi(Z | C) \| p_\theta(Z | C)). \end{aligned} \quad (\text{A.3})$$

Eq. (A.3) clarifies the probabilistic role of each of the three modules: the encoder determines what information the latent carries from the target text, the latent generator fits the conditional latent distribution induced by the encoder, and the decoder models tokens conditioned on the latent. Flow Matching is merely the particular solver used in this work to learn $p_\theta(Z | C)$, not part of the definition of the generative model itself.

A.2 Information Identities and Semantic Decomposition

Conditional mutual information is defined as

$$I(Y; Z | C) = H(Y | C) - H(Y | C, Z), \quad (\text{A.4})$$

and rearranging directly yields Eq. (3.5) in the main text.

If S is a high-level semantic variable determined by Y , then $I(Y; Z | C) = I(S, Y; Z | C)$. Applying the chain rule of mutual information,

$$\begin{aligned} I(Y; Z | C) &= I(S, Y; Z | C) \\ &= I(S; Z | C) + I(Y; Z | S, C), \end{aligned} \quad (\text{A.5})$$

which yields the semantic decomposition in Eq. (3.6). Under the compression constraint $K = \rho T$, we further define the semantic information density per latent position as

$$\mathcal{D}_{\text{sem}} = \frac{I(S; Z | C)}{K}. \quad (\text{A.6})$$

This quantity need not be estimated explicitly during training; it distinguishes two different objectives: merely shortening the latent sequence versus concentrating semantic information in a limited number of latent positions.

If the target tokens are partitioned into a visible part Y^{vis} and a target part Y^{tar} , then

$$\begin{aligned} I(Y^{\text{tar}}; Z | C, Y^{\text{vis}}) &= H(Y^{\text{tar}} | C, Y^{\text{vis}}) \\ &\quad - H(Y^{\text{tar}} | C, Y^{\text{vis}}, Z). \end{aligned} \quad (\text{A.7})$$

Generation-oriented latent supervision can therefore be interpreted as biasing the limited latent capacity toward predictive semantic information by reducing the conditional uncertainty of missing or future tokens given the latent. Section B gives a concrete interpretation of the fixed-span encoder-mask objective.

A.3 KL, Compression Rate, and Latent Information Capacity

Let $p_0(Z | C)$ be the reference distribution used for regularization in Stage 1. For any such reference distribution,

$$\begin{aligned} & \mathbb{E}_{C,Y} \text{KL}(q_\phi(Z | C, Y) \| p_0(Z | C)) \\ &= I(Y; Z | C) + \mathbb{E}_C \text{KL}(\bar{q}_\phi(Z | C) \| p_0(Z | C)) \\ &\geq I(Y; Z | C). \end{aligned} \tag{A.8}$$

Thus, the VAE KL term not only constrains the latent distribution but also upper-bounds the amount of information the latent can carry. If the posterior approximately factorizes across latents and dimensions, and the average KL per latent dimension stays roughly at $\bar{r}$, then the total KL scales approximately with Kd_z . Since $K = \rho T$, the information capacity available per token follows the approximate scaling

$$\frac{1}{T} I(Y; Z | C) \lesssim \rho d_z \bar{r}. \tag{A.9}$$

Cola2 normalizes the KL per latent element so that the regularization pressure on each latent dimension remains as comparable as possible across different K and d_z . Changing ρ can therefore be understood more directly as changing the number of latent channels available per token. Dynamic segmentation further determines where these limited latent positions are allocated under a fixed target compression rate: the compression rate controls *how many* representation positions exist, and semantic boundaries determine *where* they are placed.

B Information-Theoretic Interpretation of Generation-Oriented Latent Supervision

The $\mathcal{L}_{\text{gen}}$ term in the main text is not an additional inference-time sampler, but a fixed-span encoder-mask objective applied directly to the encoder. Its constraint is that a supervised token cannot simply rely on a complete current latent observation in one-to-one correspondence with itself, but must be predicted from a context in which its input is missing. The objective therefore specifically increases the *predictive* information of the latent rather than merely its local reconstructability.

B.1 Fixed-Span Encoder-Mask / BERT-Style Objective

Let M denote a randomly selected set of spans, Y_M the text in these spans that is masked on the encoder side, $Y_{\bar{M}}$ the remaining visible text, and Z^M the latent obtained by encoding the masked input under the fixed original span layout. Since the encoder cannot observe the ground-truth Y_M in this forward pass, we have, given (C, M) , the Markov chain

$$Y_M \longrightarrow (C, Y_{\bar{M}}) \longrightarrow Z^M. \tag{B.1}$$

By the data processing inequality,

$$I(Y_M; Z^M | C, M) \leq I(Y_M; Y_{\bar{M}} | C, M). \tag{B.2}$$

Hence the objective cannot be satisfied by directly copying the masked tokens; it can only exploit information in the visible context that is statistically related to the target spans. For any predictor $p_\eta(Y_M | C, Z^M, M)$, the cross-entropy can be written as

$$\begin{aligned} \mathcal{L}_{\text{gen}}^{\text{mask}} &= H(Y_M | C, Z^M, M) \\ &\quad + \mathbb{E} \text{KL}(q(Y_M | C, Z^M, M) \| p_\eta(Y_M | C, Z^M, M)). \end{aligned} \tag{B.3}$$

If the predictor is conditionally optimal, then

$$\begin{aligned} (\mathcal{L}_{\text{gen}}^{\text{mask}})^* &= H(Y_M | C, Z^M, M) \\ &= H(Y_M | C, M) - I(Y_M; Z^M | C, M). \end{aligned} \tag{B.4}$$

Therefore, when the encoder is trainable, reducing this objective drives the latent to contain more predictive information about the missing spans. The current MDM implementation uses fixed-span encoder-mask MDM: the first clean forward pass determines the hard spans, and the second pass replaces only the encoder observations without re-segmenting. Functionally, it thus imposes BERT-style generation pressure, but it is not equivalent to arbitrary forms of the span-BERT loss used in earlier implementations.

C Latent Hacking and Gradient Decoupling in Compressed Representations

This appendix proves Lemma 4.2, and first distinguishes two kinds of “joint training.” We deliberately retain the gradient from the discrete generator decoder to the encoder, because token reconstruction and generation are precisely the discrete realization that the latent must ultimately serve. The risk discussed here refers only to allowing the DiT Flow Matching prior loss to backpropagate into the encoder. That loss optimizes how easily the current latent can be fit by the prior solver, which is not equivalent to the reconstructability or generability of the discrete semantics. Below, we first analyze the potential conflict between the two gradients, and then discuss representation drift under fixed capacity and local decoder geometry. These analyses do not require compression to necessarily shrink the valid region, nor do they constitute a general guarantee of training collapse.

C.1 Joint Objective and Gradient Conflict

Collect the Stage 1 objectives that shape the representation into

$$\mathcal{L}_{\text{rep}}(\phi, \psi) = \mathcal{L}_{\text{bal}} + \lambda_{\text{gen}}\mathcal{L}_{\text{gen}} + \lambda_{\text{dur}}\mathcal{L}_{\text{dur}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}, \quad (\text{C.1})$$

where ϕ and ψ denote the encoder and decoder parameters, respectively; both $\mathcal{L}_{\text{bal}}$ and $\mathcal{L}_{\text{gen}}$ update the encoder through the token-prediction path of the discrete generator decoder. If the gradient of the DiT Flow Matching loss is fully allowed to reach the encoder, the encoder effectively optimizes

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{rep}} + \lambda_{\text{FM}}\mathcal{L}_{\text{FM}}. \quad (\text{C.2})$$

Define

$$g_{\text{rep}} = \nabla_{\phi}\mathcal{L}_{\text{rep}}, \quad g_{\text{FM}} = \nabla_{\phi}\mathcal{L}_{\text{FM}}. \quad (\text{C.3})$$

A single SGD update is

$$\phi^+ = \phi - \eta(g_{\text{rep}} + \lambda_{\text{FM}}g_{\text{FM}}). \quad (\text{C.4})$$

A first-order expansion of $\mathcal{L}_{\text{rep}}$ gives

$$\Delta\mathcal{L}_{\text{rep}} \approx -\eta\|g_{\text{rep}}\|_2^2 - \eta\lambda_{\text{FM}}\langle g_{\text{rep}}, g_{\text{FM}} \rangle. \quad (\text{C.5})$$

Therefore, whenever

$$\langle g_{\text{rep}}, g_{\text{FM}} \rangle < -\frac{\|g_{\text{rep}}\|_2^2}{\lambda_{\text{FM}}}, \quad (\text{C.6})$$

the DiT loss can decrease while the representation objective locally increases. This is the most basic condition for unstable joint training: the two gradients are not inherently aligned.

C.2 The DiT-to-Encoder Gradient Is Not a Semantic Gradient

Let the encoder produce the clean latent via reparameterization,

$$z_0 = g_{\phi}(x, \epsilon_0), \quad \epsilon_0 \sim \mathcal{N}(0, I). \quad (\text{C.7})$$

A general v-prediction / Flow Matching process can be written as

$$z_t = \alpha_t z_0 + \sigma_t \epsilon, \quad v_t^* = a_t z_0 + b_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad (\text{C.8})$$

and we define

$$r = v_{\theta}(z_t, t, c) - v_t^*, \quad \ell_v = \|r\|_2^2, \quad (\text{C.9})$$

where c denotes conditions such as previous latent blocks or the prompt representation. If c is detached from z_0 , the chain rule gives

$$\nabla_{z_0} \ell_v = 2(\alpha_t J_{v,z_t}^\top - a_t I) r, \quad (\text{C.10})$$

where $J_{v,z_t} = \partial v_\theta / \partial z_t$. If the condition path also depends on z_0 and is not detached, an additional term appears:

$$2J_{c,z_0}^\top J_{v,c}^\top r. \quad (\text{C.11})$$

Finally,

$$\nabla_\phi \ell_v = (\nabla_{z_0} \ell_v)^\top \frac{\partial z_0}{\partial \phi}. \quad (\text{C.12})$$

These gradients are determined jointly by the denoising residual, the DiT Jacobian, the condition path, and the noise schedule; they contain neither a reconstruction likelihood nor a semantic objective. Therefore, g_{FM} serves to make the current latent representation easier for the prior solver to fit, rather than to guarantee that the decodable semantics of the latent remain intact.

C.3 Lowering the FM Loss Admits a “Simpler Latent” Shortcut

Let $W = (z_t, t, c)$ and let the target velocity be u_t . For any vector field $v(W)$, the squared loss admits the conditional-expectation decomposition

$$\begin{aligned} \mathbb{E} \|v(W) - u_t\|_2^2 &= \mathbb{E} \|v(W) - \mathbb{E}[u_t | W]\|_2^2 \\ &\quad + \mathbb{E} \|u_t - \mathbb{E}[u_t | W]\|_2^2. \end{aligned} \quad (\text{C.13})$$

Hence

$$v^*(W) = \mathbb{E}[u_t | W], \quad \mathcal{L}_{\text{FM}}^* = \mathbb{E}[\text{Var}(u_t | z_t, t, c)]. \quad (\text{C.14})$$

Eq. (C.14) shows that, when the encoder is also trainable, the FM loss can decrease in two ways: the DiT can better approximate the conditional mean, or the encoder can actively alter the latent geometry so that the conditional variance of the target velocity itself decreases. The latter path directly flattens semantically meaningful modes.

As a simple counterexample, let the semantic variable $S \in \{-1, +1\}$ be equiprobable and construct

$$q_{\text{sem}}(z | S) = \mathcal{N}(mS e_1, I), \quad m > 0, \quad q_{\text{col}}(z | S) = \mathcal{N}(0, I). \quad (\text{C.15})$$

The former satisfies $I(S; Z_{\text{sem}}) > 0$, whereas the latter satisfies $I(S; Z_{\text{col}}) = 0$. However, in high-noise or weak-conditioning regions, transporting the bimodal semantic distribution is more complex, while the collapsed distribution is closer to the base Gaussian. It is therefore possible that

$$\mathcal{L}_{\text{FM}}(q_{\text{col}}) < \mathcal{L}_{\text{FM}}(q_{\text{sem}}), \quad I(S; Z_{\text{col}}) < I(S; Z_{\text{sem}}). \quad (\text{C.16})$$

Consequently, a lower FM loss alone cannot rule out semantic collapse; under unrestricted joint training, collapse may even become an easier optimization direction.

C.4 Information Constraints and Local Decoder Geometry under Compression

In this subsection, n denotes the number of tokens covered by a single latent Z , to distinguish it from the latent sequence length K used elsewhere. Given the data distribution and context C , define the minimum conditional information required to reach a total reconstruction-loss threshold ε as

$$\begin{aligned} R_n^*(\varepsilon) &= \inf_{q(z|X_{1:n}, C)} \left\{ I(X_{1:n}; Z | C) : \right. \\ &\quad \left. \inf_{p_\psi} \mathbb{E}[-\log p_\psi(X_{1:n} | Z, C)] \leq \varepsilon \right\}. \end{aligned} \quad (\text{C.17})$$

If the effective conditional information capacity of this latent is C_n , the information slack is

$$\Delta_n = C_n - R_n^*(\varepsilon). \quad (\text{C.18})$$

If additional tokens increase the required predictive information while C_n does not increase accordingly, the slack may decrease. This depends on the data, the context, and the distortion criterion, and does not imply an unconditional monotonic relation among separately retrained models with different compression rates.

To illustrate the geometric constraint, fix the decoder, the target span $x_{1:n}$, and the context c . Under the causal factorization, let $p_i(z) = p_\psi(x_i | x_{<i}, z, c)$, and let $\tau, \tau_i \in (0, 1]$ denote probability thresholds. Define

$$\mathcal{A}_{1:n}(\tau) = \left\{ z : \prod_{i=1}^n p_i(z) \geq \tau \right\}, \quad \mathcal{A}_i(\tau_i) = \{z : p_i(z) \geq \tau_i\}. \quad (\text{C.19})$$

Satisfying every per-token threshold is sufficient for satisfying the joint threshold, so

$$\mathcal{B}_n := \bigcap_{i=1}^n \mathcal{A}_i(\tau_i) \subseteq \mathcal{A}_{1:n} \left(\prod_{i=1}^n \tau_i \right). \quad (\text{C.20})$$

With the existing p_i and thresholds unchanged, adding one token constraint gives

$$\mathcal{B}_{n+1} = \mathcal{B}_n \cap \mathcal{A}_{n+1}(\tau_{n+1}) \subseteq \mathcal{B}_n. \quad (\text{C.21})$$

This only states that the intersection of fixed constraints does not grow; it does not guarantee strict shrinkage, nor does it directly compare relearned latent geometries.

To exclude the trivial scale growth caused by summing losses, we use the per-token average loss below. With $\ell_i(z) = -\log p_i(z)$,

$$\mathcal{L}_{\text{dec}}^{(n)}(z) = \frac{1}{n} \sum_{i=1}^n \ell_i(z). \quad (\text{C.22})$$

Its local Hessian is

$$H_n(z) = \nabla_z^2 \mathcal{L}_{\text{dec}}^{(n)}(z) = \frac{1}{n} \sum_{i=1}^n \nabla_z^2 \ell_i(z). \quad (\text{C.23})$$

For MDM, the same local analysis can take ℓ_i as the token denoising loss under a fixed masked state, without treating the product as the full MDM sequence likelihood. If the loss is twice continuously differentiable near z , then for a latent drift δ ,

$$\begin{aligned} \mathcal{L}_{\text{dec}}^{(n)}(z + \delta) - \mathcal{L}_{\text{dec}}^{(n)}(z) &= \nabla \mathcal{L}_{\text{dec}}^{(n)}(z)^\top \delta + \frac{1}{2} \delta^\top H_n(z) \delta \\ &\quad + o(\|\delta\|_2^2). \end{aligned} \quad (\text{C.24})$$

Only when z is close to a stationary point, the first-order and higher-order terms are negligible, and $\|H_n(z)\|_2 > 0$, the local quadratic perturbation scale corresponding to an allowed average-loss increase $\epsilon_d > 0$ is approximately

$$r_n^{\text{quad}}(\epsilon_d) \simeq \sqrt{\frac{2\epsilon_d}{\|H_n(z)\|_2}}. \quad (\text{C.25})$$

This scale is neither the boundary of the actual valid region nor a guaranteed safe radius. More tokens need not increase the spectral norm of the average Hessian, and the actual loss change also depends on the drift direction. Only when the additional constraints increase the local curvature along the relevant directions can the decoder become more sensitive to drift along those directions.

C.5 Prior-Induced Local Drift and Gradient Decoupling

Considering only the DiT gradient component acting on the encoder, one parameter update is

$$\delta\phi_{\text{FM}} = -\eta\lambda_{\text{FM}}g_{\text{FM}}. \quad (\text{C.26})$$

Let $J_{z,\phi} = \partial z_0 / \partial \phi$ be the encoder Jacobian. Taking $g_{\text{FM}} = \nabla_{\phi} \ell_v$ for the single-sample FM term, the induced first-order latent drift is

$$\begin{aligned} \delta z_{\text{FM}} &\approx J_{z,\phi} \delta \phi_{\text{FM}} \\ &= -\eta \lambda_{\text{FM}} J_{z,\phi} J_{z,\phi}^{\top} \nabla_{z_0} \ell_v. \end{aligned} \quad (\text{C.27})$$

For batch updates, g_{FM} should be replaced by the corresponding aggregated parameter gradient. Fixing the current decoder and setting $z = z_0$, Eq. (C.24) gives

$$\begin{aligned} \Delta \mathcal{L}_{\text{dec}}^{\text{FM}} &:= \mathcal{L}_{\text{dec}}^{(n)}(z + \delta z_{\text{FM}}) - \mathcal{L}_{\text{dec}}^{(n)}(z) \\ &\approx \nabla \mathcal{L}_{\text{dec}}^{(n)}(z)^{\top} \delta z_{\text{FM}} + \frac{1}{2} \delta z_{\text{FM}}^{\top} H_n(z) \delta z_{\text{FM}}. \end{aligned} \quad (\text{C.28})$$

Hence, the FM-induced drift may increase the token loss along sensitive directions of the decoder, with an effect that depends on the encoder Jacobian, the direction and magnitude of the drift, and the simultaneous adaptation of the decoder. This analysis characterizes the local risk of prior-induced representation drift; it does not establish a general relation between the compression rate and training collapse.

Full detachment directly removes the prior-solver gradient acting on the encoder:

$$\nabla_{\phi} \mathcal{L}_{\text{joint}} \xrightarrow{\text{full detach}} \nabla_{\phi} \mathcal{L}_{\text{rep}}, \quad \nabla_{\theta} \mathcal{L}_{\text{joint}} = \lambda_{\text{FM}} \nabla_{\theta} \mathcal{L}_{\text{FM}}. \quad (\text{C.29})$$

The VAE still learns under its own objectives, and the DiT still fits the aggregated posterior using samples from the current encoder. This operation guarantees that the path by which the prior loss directly rewrites the encoder is blocked, but does not guarantee that the full training process is necessarily stable.

D Joint Information-Theoretic Interpretation of VAE and DiT Noise

This appendix proves Lemma 4.3. To analyze the relation between the Stage 1 latent noise and the Stage 2 diffusion noise, we write the clean latent in the simplified form

$$Z = G + N_0, \quad (\text{D.1})$$

where G denotes the signal associated with high-level semantics and N_0 denotes the VAE posterior noise. The DiT further constructs

$$Z_t = \alpha_t Z + \sigma_t \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I). \quad (\text{D.2})$$

Under a Gaussian approximation, assume that $G \sim \mathcal{N}(0, \Sigma_G)$ and $N_0 \sim \mathcal{N}(0, \Sigma_0)$ are independent. Conditioned on G , the effective noise covariance of Z_t is $\alpha_t^2 \Sigma_0 + \sigma_t^2 I$, and therefore

$$I(G; Z_t) = \frac{1}{2} \log \det [I + \alpha_t^2 \Sigma_G (\alpha_t^2 \Sigma_0 + \sigma_t^2 I)^{-1}]. \quad (\text{D.3})$$

Defining $\lambda_t = \log(\alpha_t^2 / \sigma_t^2)$, the same expression can equivalently be written as

$$I(G; Z_t) = \frac{1}{2} \log \det [I + \Sigma_G (\Sigma_0 + e^{-\lambda_t} I)^{-1}]. \quad (\text{D.4})$$

Eq. (D.4) expresses a central fact: the VAE posterior noise and the DiT corruption jointly determine how discriminable the latent semantic signal is. Applying the same DiT noise schedule to VAE latents of different sharpness therefore corresponds to different semantic-information regimes. We accordingly treat the two noise sources as quantities that must be calibrated jointly, rather than claiming that any fixed logSNR or timestep is a model-independent theoretical optimum.

E Reconstruction--Generation Balance in the Discrete Generator Decoder

This appendix proves Lemma 4.4.

E.1 Information Allocation under Different Latent Conditions

Let Z_A denote the current-block latent condition actually received by the discrete generator decoder under condition mode A , where $A \in \{\text{cond}, \text{noise}, \text{drop}\}$. For each condition, Eq. (3.5) in the main text holds:

$$H(Y | C) = I(Y; Z_A | C) + H(Y | C, Z_A). \quad (\text{E.1})$$

The **cond** state uses the complete clean latent and therefore typically provides the most informative condition for the current block. The **drop** state zeros the latent condition of the current block, which corresponds to masked token prediction that does not depend on that latent. The **noise** state applies a small random perturbation to the clean latent to train the decoder’s robustness to imprecise conditions. Together, the three states cover three training regimes: strongly conditioned reconstruction, local robustness, and weakly conditioned generation.

For any discrete generator decoder $p_\psi(Y | C, Z_A)$, the cross-entropy decomposes exactly as

$$\begin{aligned} \mathcal{L}_A(\psi) &= H(Y | C, Z_A) \\ &\quad + \mathbb{E} \text{KL}(q(Y | C, Z_A) \| p_\psi(Y | C, Z_A)). \end{aligned} \quad (\text{E.2})$$

The first term is determined by how much uncertainty remains under the given condition; the second is the modeling error of the finite-capacity discrete generator decoder relative to the optimal conditional distribution. Training only under **cond** mainly optimizes the strongly conditioned region and may make the model overly dependent on precise latents, whereas training only under **drop** may cause the discrete generator decoder to ignore the latent. Mixing **cond**, **noise**, and **drop** trains the same decoder parameters to reduce the modeling error across different levels of condition reliability. The optimal mixture depends on model capacity, training budget, and the distribution of latent errors at inference, and should therefore be determined empirically.

E.2 Local Perturbation Path of the Noise Condition

The **noise** condition uses the same linear interpolation as the latent generator, but restricts the range of the perturbation coefficient. Let $u \in [0, u_{\max}]$, where $u_{\max} \ll 1$ controls the perturbation scale; then

$$Z_u = (1 - u)Z + u\varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I). \quad (\text{E.3})$$

For a single d_z -dimensional latent, let $\Delta Z_u = Z_u - Z$. Independent standard Gaussian noise gives

$$\mathbb{E}[\|\Delta Z_u\|_2^2 | Z, u] = u^2(d_z + \|Z\|_2^2). \quad (\text{E.4})$$

A small u controls the mean-squared perturbation scale, but Gaussian noise does not have finite support. For each u , the decoder receives information $I(Y; Z_u | C)$; if a stronger perturbation constitutes a more degraded stochastic channel, this mutual information does not increase, and the corresponding $H(Y | C, Z_u)$ does not decrease. The **noise** branch optimizes the expected token loss under this perturbation distribution to improve adaptation to imprecise latents; it neither assumes that the artificial perturbation shares the distribution of DiT output errors, nor guarantees uniform robustness to all inference-time deviations.

F Probabilistic Derivation of Classifier-Free Guidance

The single-axis guidance of the latent DiT and each axis of the two-axis MDM guidance are based on the linear operator

$$f_\gamma = f_u + \gamma(f_c - f_u), \quad (\text{F.1})$$

where f_c and f_u are the branch outputs with and without the selected condition, and γ is the guidance scale; the latent generator applies this operator to velocities, whereas the MDM decoder combines logits along the prefix and latent axes. Below, we give a per-noise-time interpretation under exact scores and optimal FM regression fields, and derive the token-probability reweighting under a fixed decoding state; neither automatically yields a power-mixture guarantee for the complete generated distribution.

F.1 From Bayes' Rule to the Guided Score

First consider a continuous variable X_u with a differentiable positive density, where u is the noise time and x its value. Let $\mathcal{K}$ denote the context shared by the conditional and unconditional branches, let A denote the additional condition dropped in CFG, and define

$$\begin{aligned} s_c(x, u) &= \nabla_x \log p_u(x \mid \mathcal{K}, A), \\ s_u(x, u) &= \nabla_x \log p_u(x \mid \mathcal{K}). \end{aligned} \quad (\text{F.2})$$

By Bayes' rule,

$$p_u(x \mid \mathcal{K}, A) = \frac{p_u(A \mid x, \mathcal{K}) p_u(x \mid \mathcal{K})}{p(A \mid \mathcal{K})}, \quad (\text{F.3})$$

and taking the gradient with respect to x yields

$$s_c(x, u) - s_u(x, u) = \nabla_x \log p_u(A \mid x, \mathcal{K}). \quad (\text{F.4})$$

The standard CFG score therefore satisfies

$$\begin{aligned} s_\gamma &= s_u + \gamma(s_c - s_u) \\ &= \nabla_x \log p_u(x \mid \mathcal{K}) + \gamma \nabla_x \log p_u(A \mid x, \mathcal{K}) \\ &= \nabla_x \log \tilde{p}_{\gamma, u}(x \mid \mathcal{K}, A), \end{aligned} \quad (\text{F.5})$$

where the unnormalized guided distribution is

$$\begin{aligned} \tilde{p}_{\gamma, u}(x \mid \mathcal{K}, A) &\propto p_u(x \mid \mathcal{K}) p_u(A \mid x, \mathcal{K})^\gamma \\ &\propto p_u(x \mid \mathcal{K}, A)^\gamma p_u(x \mid \mathcal{K})^{1-\gamma}. \end{aligned} \quad (\text{F.6})$$

When the normalizing constant is finite, it is independent of x and does not affect the score. For a fixed u , in Eq. (F.6), $\gamma = 0$ recovers the base density at that time, $\gamma = 1$ recovers the conditional density, $0 < \gamma < 1$ attenuates the conditional likelihood ratio, and $\gamma > 1$ amplifies it. This equivalence describes the instantaneous score field rather than the final sampling distribution.

F.2 Equivalence of Flow Matching Velocity CFG and Score CFG

The latent generator of *Cola2* predicts velocities rather than scores, so it remains to explain why applying the same linear combination to velocities, as in Eq. (3.36), is still valid. Consider a general Gaussian corruption path

$$X_u = \alpha_u X_0 + \sigma_u \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I), \quad (\text{F.7})$$

where α_u and σ_u are differentiable path coefficients and dots denote derivatives with respect to the noise time u . The following identities apply at interior times with $\alpha_u \neq 0$ and $\sigma_u > 0$. For an arbitrary condition Q , write

$$s_Q(x, u) = \nabla_x \log p_u(x \mid Q). \quad (\text{F.8})$$

The score identity for Gaussian convolution gives

$$\mathbb{E}[\varepsilon \mid X_u = x, Q] = -\sigma_u s_Q(x, u). \quad (\text{F.9})$$

On the other hand, since $x = \alpha_u X_0 + \sigma_u \varepsilon$,

$$\mathbb{E}[X_0 \mid X_u = x, Q] = \frac{x + \sigma_u^2 s_Q(x, u)}{\alpha_u}. \quad (\text{F.10})$$

The optimal Flow Matching regression target is the conditional expected path velocity, so

$$\begin{aligned} v_Q(x, u) &= \mathbb{E}[\dot{\alpha}_u X_0 + \dot{\sigma}_u \varepsilon \mid X_u = x, Q] \\ &= \frac{\dot{\alpha}_u}{\alpha_u} x + \sigma_u \left(\frac{\dot{\alpha}_u \sigma_u}{\alpha_u} - \dot{\sigma}_u \right) s_Q(x, u). \end{aligned} \quad (\text{F.11})$$

In other words, at a fixed u , the velocity and the score are related by an affine transformation shared by all conditions,

$$v_Q(x, u) = a_u x + b_u s_Q(x, u), \quad (\text{F.12})$$

where $a_u = \dot{\alpha}_u / \alpha_u$ and $b_u = \sigma_u (\dot{\alpha}_u \sigma_u / \alpha_u - \dot{\sigma}_u)$. When the conditional and unconditional branches are subtracted, the shared term $a_u x$ cancels exactly, so

$$\begin{aligned} v_u + \gamma_{\text{gen}}(v_c - v_u) &= a_u x + b_u [s_u + \gamma_{\text{gen}}(s_c - s_u)] \\ &= a_u x + b_u s_\gamma. \end{aligned} \quad (\text{F.13})$$

Therefore, under a shared Gaussian path and optimal branch regression fields, velocity CFG and score CFG correspond pointwise, with $\gamma = \gamma_{\text{gen}}$. For the linear path used in this work,

$$\alpha_u = 1 - u, \quad \sigma_u = u, \quad (\text{F.14})$$

Eq. (F.11) further reduces to

$$v_Q(x, u) = -\frac{x}{1-u} - \frac{u}{1-u} s_Q(x, u), \quad 0 < u < 1, \quad (\text{F.15})$$

with the endpoints understood as continuous limits when these exist. In *Cola2*, v_c uses the historical latent cache while v_u clears it, so γ_{gen} modulates the influence of the historical condition on the instantaneous field.

Let $v_\gamma = v_u + \gamma(v_c - v_u)$. If the normalized density in Eq. (F.6) were also the actual marginal of this flow, it would have to satisfy the continuity equation

$$\partial_u \tilde{p}_{\gamma, u} + \nabla_x \cdot (\tilde{p}_{\gamma, u} v_\gamma) = 0. \quad (\text{F.16})$$

The per-time score identity alone does not guarantee this path-consistency condition. Hence, even with exact branch fields, one cannot conclude that the endpoint of guided integration follows the data distribution reweighted by the same power.

F.3 Prefix--Latent Two-Axis CFG for MDM

For the `masked_only` MDM, let $\mathcal{P}$ be the committed pure-token prefix, $\mathcal{U}$ the token/mask state of the current block together with the fixed span layout, and H^Z the current latent condition refined by the condition stack. Denoting the decoder logits by $\ell(y \mid \cdot)$, the four branches are

$$\begin{aligned} \ell_{11}(y) &= \ell(y \mid \mathcal{U}, \mathcal{P}, H^Z), & \ell_{10}(y) &= \ell(y \mid \mathcal{U}, \mathcal{P}, 0), \\ \ell_{01}(y) &= \ell(y \mid \mathcal{U}, \emptyset, H^Z), & \ell_{00}(y) &= \ell(y \mid \mathcal{U}, \emptyset, 0). \end{aligned} \quad (\text{F.17})$$

The first subscript controls the visibility of the prefix token KV, and the second controls the additive injection of the current latent; 0 is a zero condition tensor rather than a learnable null embedding. All four branches share $\mathcal{U}$ and the same H^Z , so ℓ_{00} can still read the known tokens of the current block and is not an uninformed unconditional text model.

Using guidance scales γ_p and γ_z for the prefix and latent axes, respectively, the expansion is

$$\begin{aligned} \ell_{\gamma_p, \gamma_z} &= \gamma_p \gamma_z \ell_{11} + \gamma_p (1 - \gamma_z) \ell_{10} \\ &\quad + (1 - \gamma_p) \gamma_z \ell_{01} + (1 - \gamma_p) (1 - \gamma_z) \ell_{00}. \end{aligned} \quad (\text{F.18})$$

This bilinear combination is independent of the nesting order of the two axes. Setting $\gamma_p = 1$ modulates only the latent, $\gamma_z = 1$ modulates only the prefix, and $\gamma_z = 0$ does not use the current latent condition. In general, dropping nonzero terms no longer yields this two-axis rule.

Let c_{ab} be the coefficient of ℓ_{ab} in Eq. (F.18), and $p_{ab} = \text{softmax}(\ell_{ab})$ the corresponding token probability. For any candidate token y ,

$$\ell_{ab}(y) = \log p_{ab}(y) + \kappa_{ab}, \quad (\text{F.19})$$

where κ_{ab} is the log-partition constant of each branch and is independent of y . For a fixed current state,

$$\sum_{a,b \in \{0,1\}} c_{ab} = 1, \quad p_{\gamma_p, \gamma_z}(y) \propto \prod_{a,b \in \{0,1\}} p_{ab}(y)^{c_{ab}}. \quad (\text{F.20})$$

This algebraic identity does not require the network to reach the true conditional distribution. The coefficients can be negative, so this is an affine combination of logits rather than a convex mixture of probabilities; coefficients summing to 1 also do not guarantee that the sharpness of the distribution or the confidence ranking used for demasking is preserved. Prefix-off only masks the token-cache read path and does not remove the historical information retained in H^Z , so these operational branches are not directly equated with four marginal conditions of the true joint distribution.

This rule acts on the token distribution of each prediction; since the states and normalizers of multi-step demasking change with the generated results, a global power mixture of the complete text distribution does not follow automatically. The key mask, position indexing, and read-only cache protocol are given in Appendix G.6.1.

G Implementation Details of the Overall Workflow

This appendix supplements the dynamic segmentation, the block-causal MDM decoder, the Stage 1/Stage 2 gradient routing, and the complete inference procedure with latent feedback omitted from Section 3.2. The main text retains only the core definitions needed for unified modeling; implementation details are expanded here.

G.1 Dynamic Semantic Segmentation and Compression-Rate Control

Let $h_t = \text{Emb}(y_t)$ denote the token representation before segmentation. The semantic discontinuity between adjacent positions is defined as

$$\delta_t = 1 - \cos(\text{LN}(h_t), \text{LN}(h_{t+1})), \quad t = 1, \dots, T-1, \quad (\text{G.1})$$

and is mapped to a boundary probability by

$$p_t = \sigma(\alpha(\delta_t - \tau_0) + b), \quad \text{boundary}_t = \mathbb{1}[p_t > 0.5], \quad (\text{G.2})$$

where α is the slope, τ_0 the threshold, and b a scalar bias controlling the overall boundary density. If the target compression rate is ρ^* and the measured rate on the current global batch is $\hat{\rho}$, then b does not participate in backpropagation and is updated as

$$b \leftarrow \text{clip}(b - \eta_b(\hat{\rho} - \rho^*), -c_b, c_b), \quad (\text{G.3})$$

where η_b is the control step size and c_b the clipping range. The upper bound $L_{\max}$ is enforced by forced cuts during a greedy scan.

G.1.1 Convergence of Compression-Rate Control

By Eq. (G.2), a hard boundary satisfies

$$p_t > \frac{1}{2} \iff \delta_t > \tau_{\text{eff}}(b) := \tau_0 - \frac{b}{\alpha}. \quad (\text{G.4})$$

Thus, increasing b lowers the effective threshold and increases the boundary density; together with Eq. (G.3), the sign of b forms a negative-feedback loop. With the encoder and data distribution fixed, define

$$r(b) := \mathbb{E}[\hat{\rho} \mid b] \quad (\text{G.5})$$

as the expected compression rate after hard thresholding and length constraints. Length-based forced cuts introduce plateaus and an attainable lower bound into $r(b)$, but as long as the target ρ^* lies within the attainable range and the local rate response near the equilibrium remains monotone, the following stability result holds.

Lemma G.1 (Local Convergence of Dynamic Compression-Rate Control). *Assume that there exists $b^* \in (-c_b, c_b)$ such that $r(b^*) = \rho^*$. If r is differentiable in a neighborhood $\mathcal{I}$ of b^* and satisfies*

$$0 < m \leq r'(b) \leq L < \infty, \quad b \in \mathcal{I}, \quad (\text{G.6})$$

then for any $0 < \eta_b < 2/L$, the mean-response update map

$$F(b) = \Pi_{[-c_b, c_b]}[b - \eta_b(r(b) - \rho^*)] \quad (\text{G.7})$$

is a contraction in a sufficiently small neighborhood of b^* , with

$$|b_{n+1} - b^*| \leq q|b_n - b^*|, \quad q = \max\{|1 - \eta_b m|, |1 - \eta_b L|\} < 1. \quad (\text{G.8})$$

Consequently, $b_n \rightarrow b^*$ and $r(b_n) \rightarrow \rho^*$, with at least a geometric convergence rate $\mathcal{O}(q^n)$.

Proof. Let $e_n = b_n - b^*$. By the mean value theorem, there exists ξ_n between b_n and b^* such that

$$r(b_n) - r(b^*) = r'(\xi_n)e_n. \quad (\text{G.9})$$

Ignoring the projection, the mean-response update in Eq. (G.7) gives

$$e_{n+1} = (1 - \eta_b r'(\xi_n))e_n. \quad (\text{G.10})$$

From Eq. (G.6) and $0 < \eta_b < 2/L$, we have $|1 - \eta_b r'(\xi_n)| \leq q < 1$. The interval projection $\Pi_{[-c_b, c_b]}$ is non-expansive, and since b^* lies in the interior of the interval, the projection does not enlarge the error. This yields Eq. (G.8) and completes the proof. $\square$

In practice, with the encoder fixed, the hard rate on a finite global batch can be written as $\hat{\rho}_n = r(b_n) + \varepsilon_n$, where ε_n denotes measurement fluctuation. If the i -th sample contains T_i valid tokens, the quantization step of the rate is approximately $1/\sum_i T_i$. Under local stability and controlled noise, a fixed η_b typically results in fluctuation around the target; using a decreasing step size with $\sum_n \eta_n = \infty$ and $\sum_n \eta_n^2 < \infty$ additionally requires stochastic-approximation conditions such as zero-mean bounded-variance noise and iterate stability before convergence can be discussed.

During joint training, let ϕ_n be the encoder parameters at the n -th update, $r_n(b) = \mathbb{E}[\hat{\rho}_n | b, \phi_n]$ the actual response, and b_n^* its equilibrium satisfying $r_n(b_n^*) = \rho^*$. If the response at every step has a uniform contraction factor $q < 1$ over the visited region, then

$$|b_{n+1} - b_{n+1}^*| \leq q|b_n - b_n^*| + \eta_b|\varepsilon_n| + |b_{n+1}^* - b_n^*|, \quad (\text{G.11})$$

where $\varepsilon_n = \hat{\rho}_n - r_n(b_n)$. The last term captures the target drift caused by encoder updates. The lemma therefore analyzes local convergence under a fixed response and provides a tracking interpretation for slowly varying responses, rather than a convergence guarantee for fully non-stationary joint training.

Given the assignment map $g(t)$, span pooling uses block-diagonal bidirectional attention:

$$M_{ij}^{\text{pool}} = \mathbb{I}[g(i) = g(j)], \quad r_k = \frac{1}{L_k} \sum_{t \in S_k} \tilde{h}_t, \quad (\text{G.12})$$

where $\tilde{h} = \text{Attn}_{M^{\text{pool}}}(h)$. A causal Transformer along the latent axis then produces $\bar{r}_k$, and the posterior parameters are

$$[\mu_k, \log \sigma_k^2] = W_{\text{post}} \bar{r}_k. \quad (\text{G.13})$$

Reparameterized sampling is used during training, and the posterior mean is used at inference.

G.2 Decoder Duration and Token-Level Conditioning

Let $g_k = W_{\text{in}} z_k$. The duration head outputs a distribution over L_{max} classes,

$$p(L_k | z_k) = \text{softmax}(\text{MLP}_{\text{dur}}(g_k)). \quad (\text{G.14})$$

Given L_k , g_k is repeated L_k times and concatenated with the relative position within the span. If token t has local index j in its span of length L , then

$$h_t^Z = W_c \left[g_{g(t)} \parallel \text{MLP}_{\text{loc}} \left(\frac{j}{L}, \frac{L-j}{L}, \log L \right) \right]. \quad (\text{G.15})$$

Before being injected into the token decoder, this condition is further refined by a few block-causal conditioning layers. Consequently, multiple token positions expanded from the same latent remain distinguishable.

G.3 Block-Causal Masked-Diffusion Decoder

The main text denotes the token-side module by the discrete generator decoder D_ψ , instantiated as a block-causal MDM. It receives the committed token context and the token-aligned latent condition H^Z , and realizes continuous semantics as discrete tokens. The decoder is connected to the compression encoder through the duration/expansion interface, uses latent reliability augmentation, and is trained jointly with the detached DiT in Stage 2.

During training, the decoder uses a dual layout [clean | masked] of length $2T$. The clean half stores the ground-truth tokens, while selected positions in the masked half are replaced with a learnable [MASK] embedding. Let $\beta(i)$ denote the latent block containing token i , and let $\text{cl}(i) \in \{0, 1\}$ indicate whether the position belongs to the clean half. The attention mask is

$$\text{Allow}(i, j) = \begin{cases} \text{cl}(j) \wedge [\beta(j) \leq \beta(i)], & \text{cl}(i) = 1, \\ (\text{cl}(j) \wedge [\beta(j) < \beta(i)]) \vee (\neg \text{cl}(j) \wedge [\beta(j) = \beta(i)]), & \text{cl}(i) = 0. \end{cases} \quad (\text{G.16})$$

Hence token blocks are causal with respect to one another, while bidirectional denoising is allowed within the current masked block; a masked query cannot read the clean copy of its current block, which prevents target leakage. Under the `masked_only` configuration, the clean half receives no latent condition, while the masked half receives a clean, noisy, or zero latent condition according to the condition state. This design naturally decouples the historical clean-token cache from the latent condition and allows the conditioned and unconditioned logits to share the same prefix cache.

During training, each token block independently samples a masking ratio π_j , and the cross-entropy of each masked token is weighted by π_j^{-1} . With $\mathcal{M}_j$ denoting the masked positions in block j ,

$$\mathcal{L}_{\text{MDM}}^A = \frac{1}{|\mathcal{M}_A|} \sum_{t \in \mathcal{M}_A} \frac{-\log p_\psi(y_t | \tilde{Y}, H_A^Z)}{\pi_{\beta(t)}}, \quad A \in \{\text{cond}, \text{noise}, \text{drop}\}, \quad (\text{G.17})$$

where the supervised sets of the three states are normalized independently. The implementation also uses fully masked blocks with some probability to directly train the recovery of a complete new block from the latent and the historical blocks.

The decoder uses the encoder-mask generation objective as $\mathcal{L}_{\text{gen}}$. A clean encoder forward pass first fixes the hard spans; a random subset of spans is then selected, their encoder input tokens are replaced with encoder-side masks, and a second encoder forward pass is performed without rerunning the dynamic segmenter. The decoder uses fully masked queries for blocks touched by these spans, but supervises only the tokens actually masked on the encoder side:

$$\mathcal{L}_{\text{encmask}} = -\frac{1}{|\mathcal{M}_e|} \sum_{t \in \mathcal{M}_e} \log p_\psi(y_t | \tilde{Y}, Z^{\text{encmask}}). \quad (\text{G.18})$$

Since the span layout is fixed by stop-gradient, this branch forces the encoder to use context to form decodable latents when local observations are missing, rather than lowering the task difficulty by moving boundaries.

G.4 Stage 1: Unified Compression VAE / Discrete Generator Decoder Objective

The block-causal MDM decoder is trained end to end with the encoder in Stage 1; this is the core of the co-adaptation between continuous semantics and discrete realization in this work, as opposed to first fixing the encoder and then fitting the decoder. For each block, a clean latent Z is first constructed, and then a mutually exclusive condition state is sampled. The clean state uses Z directly; the noise state locally perturbs Z ; the drop state removes only the latent condition of the current block while keeping the same token history. A generic formulation is

$$\tilde{z}_k = \begin{cases} z_k, & A_k = \mathbf{cond}, \\ (1 - u_k^c)z_k + u_k^c \epsilon_k, & A_k = \mathbf{noise}, \\ \emptyset, & A_k = \mathbf{drop}, \end{cases} \quad u_k^c \in [0, u_{\max}], \quad \epsilon_k \sim \mathcal{N}(0, I). \quad (\text{G.19})$$

The implementation may additionally apply a weaker sample-level latent perturbation before state partitioning to improve adaptation to small latent perturbations; this is input augmentation rather than a separate reconstruction loss, and it does not assume the same distribution as the terminal error of the DiT.

Let $\mathcal{L}_{\text{tok}}^A$ denote the MDM masked denoising objective under state A . The reconstruction-generation balance is

$$\mathcal{L}_{\text{bal}} = \lambda_c \mathcal{L}_{\text{tok}}^{\mathbf{cond}} + \lambda_n \mathcal{L}_{\text{tok}}^{\mathbf{noise}} + \lambda_d \mathcal{L}_{\text{tok}}^{\mathbf{drop}}, \quad (\text{G.20})$$

where $\mathcal{L}_{\text{tok}}^A = \mathcal{L}_{\text{MDM}}^A$ as defined in Eq. (G.17).

Separately from the three-state decoder balance, we denote the objective dedicated to injecting *generative predictive information* into the encoder by

$$\mathcal{L}_{\text{gen}} = \mathcal{L}_{\text{encmask}}. \quad (\text{G.21})$$

The unified objective of the discrete generator decoder is therefore

$$\mathcal{L}_{\text{DGD}} = \mathcal{L}_{\text{bal}} + \lambda_{\text{gen}} \mathcal{L}_{\text{gen}}. \quad (\text{G.22})$$

The key target of $\mathcal{L}_{\text{gen}}$ is the latent representation: the masked-input encoder must recover the missing semantics from context and receives gradients directly from ground-truth discrete token prediction.

The Gaussian KL of each latent is

$$\text{KL}_k = \frac{1}{2} \sum_{d=1}^{d_z} (\mu_{k,d}^2 + \sigma_{k,d}^2 - 1 - \log \sigma_{k,d}^2), \quad (\text{G.23})$$

and the duration loss is

$$\mathcal{L}_{\text{dur}} = -\frac{1}{K} \sum_{k=1}^K \log p_{\psi}(L_k | z_k). \quad (\text{G.24})$$

Combined with Eq. (3.24), the total Stage 1 objective is

$$\mathcal{L}^{(1)} = \mathcal{L}_{\text{DGD}} + \lambda_{\text{dur}} \mathcal{L}_{\text{dur}} + \beta_{\text{KL}} \mathcal{L}_{\text{KL}}. \quad (\text{G.25})$$

This objective remains active in Stage 2; the transition from Stage 1 to Stage 2 is therefore a full-state handoff rather than “train the VAE, then freeze the VAE.” The discrete token losses can always shape the encoder latents through the condition path, and only the DiT prior loss is isolated by stop-gradient.

G.5 Stage 2: Detached Block-Causal Flow Matching

Stage 2 additionally trains the latent DiT on top of Eq. (G.25). Let Z be the sampled posterior latent of the current online encoder. The DiT reads only

$$\bar{Z} = \text{sg}(Z), \quad (\text{G.26})$$

so the DiT objective cannot backpropagate into the encoder; meanwhile, E_ϕ , the latent condition path, and D_ψ are still updated normally by $\mathcal{L}^{(1)}$.

The latent sequence is grouped into blocks of B latents. Each block independently samples a noise level u_j , all latents in a block share the same timestep, and the linear path in Eq. (3.28) is used. The training layout consists of clean history and noisy current blocks: every noisy query can read only strictly earlier clean latent blocks and interacts bidirectionally within its current noisy block. The attention mask can be written as

$$\text{Allow}_{\text{DiT}}(i, j) = \mathbb{1}[\beta(j) < \beta(i)] \text{cl}(j) \vee \mathbb{1}[\beta(j) = \beta(i)] \neg \text{cl}(j), \quad (\text{G.27})$$

where clean-history positions act as read-only conditions, and the current noisy block cannot read its corresponding clean target copy.

To cover inference cases in which the prompt is not aligned with latent block boundaries, training may randomly restore the first $c \in \{1, \dots, B-1\}$ latents of a noisy block to their clean values and remove them from the corresponding Flow Matching supervision. This partial-clean-prefix augmentation matches the handling of the first mixed block at inference. Finally,

$$\nabla_\phi \mathcal{L}_{\text{FM}} = 0, \quad \nabla_\theta \mathcal{L}_{\text{FM}} \neq 0, \quad (\text{G.28})$$

and

$$\mathcal{L}^{(2)} = \mathcal{L}^{(1)} + \lambda_{\text{FM}} \mathcal{L}_{\text{FM}}. \quad (\text{G.29})$$

Stage 2 can thus be understood as continuously updating the online compression encoder and discrete generator decoder while letting the DiT track their evolving latent distribution, but forbidding the DiT from shaping the encoder through the diffusion loss.

G.6 Complete Inference Procedure and Latent Feedback

Inference follows *propose-decode-re-encode-commit*. The dynamic encoder runs on the prompt only once at the beginning and uses the posterior mode. Suppose the prompt yields K_p latents with

$$K_p = Bq + r, \quad 0 \leq r < B. \quad (\text{G.30})$$

The first Bq block-aligned latents are prefilled into the persistent caches. If $r > 0$, the trailing r prompt latents and their tokens are not written into the persistent cache as a completed block; instead, they are kept as the clean prefix of the first generated block, and the remaining $B-r$ latent slots are initialized from Gaussian noise:

$$Z_j^{(0)} = \left[z_{K_p-r+1:K_p}^p \parallel \epsilon_1, \dots, \epsilon_{B-r} \right], \quad \epsilon_i \sim \mathcal{N}(0, I). \quad (\text{G.31})$$

This explains the meaning of “discard trailing remainder” in Figure 1: the trailing remainder is excluded from the *committed cache*, not permanently removed from the prompt semantics. After each DiT Euler update, the clean prompt prefix in the mixed block is restored to its original values.

Latent proposal. The DiT performs S_{gen} denoising steps on top of the committed latent prefix. With CFG,

$$\hat{v} = v_u + \gamma_{\text{gen}}(v_c - v_u), \quad (\text{G.32})$$

where the conditional branch reads the committed corrected latent cache and the unconditional branch removes this prefix. Importantly, the current proposal remains a read-only temporary state until token generation and feedback correction are complete, and is not written into the persistent DiT/condition caches.

Duration and token decoding. The proposal latents first pass through the duration/expansion path. For newly generated latent slots,

$$\hat{L}_k = \arg \max_L p_\psi(L \mid z_k), \quad (\text{G.33})$$

whereas the slots belonging to the prompt tail in the mixed first block keep the ground-truth span lengths known to the encoder. This yields the effective width and the token-aligned condition of the current token block.

The MDM decoder initializes unknown positions to [MASK], repeatedly predicts them over several refinement steps, and progressively commits them by confidence; intermediate masked states do not enter the KV cache, and only after the whole block is determined is a single clean forward pass used to commit the final tokens, yielding the finalized clean token block Y_j .

The decoder logits use the prefix-latent two-axis CFG in Eq. (3.37). The four branches share the current block state and the latent-free clean-token prefix cache; the prefix-off branches mask the historical token keys, and the latent-off branches zero the injection of the current proposal latent condition. Execution details are given below.

Fixed-span re-encoding. After the current token block is finalized, the feedback encoder does not rerun the dynamic segmenter. Instead, it directly constructs fixed span boundaries from the span lengths just determined by the duration predictor. It applies to these tokens the same span-local bidirectional pooling as the initial encoder, reads the committed encoder latent-prefix KV, obtains the posterior of the current block through the causal latent encoder, and takes its mode:

$$Z_j^{\text{corr}} = \mu_\phi(Y_j; \hat{L}_{B_j}, \mathcal{K}_{< j}^E). \quad (\text{G.34})$$

For the first mixed block, the previously saved pooled representations of the prompt tail are concatenated with those of the newly generated spans to form a complete B -latent block before running the latent encoder. Since the span boundaries are fixed, this correction changes “the latent obtained by re-encoding the generated tokens under this span layout,” rather than re-optimizing the segmentation.

Atomic commit. After feedback, the proposal latent is replaced by Z_j^{corr} , and the caches are updated according to the invariants

$$\begin{aligned} \mathcal{K}_j^{\text{DiT}} &= Z_{\leq j}^{\text{corr}}, \\ \mathcal{K}_j^{\text{cond}} &= \text{Cond}(Z_{\leq j}^{\text{corr}}), \\ \mathcal{K}_j^E &= \text{EncState}(Y_{\leq j}), \\ \mathcal{K}_j^{\text{tok}} &= Y_{\leq j}^{\text{clean}}. \end{aligned} \quad (\text{G.35})$$

That is, the DiT history, the condition history, and the encoder latent history all correspond to corrected encoder latents, while the token cache contains only the final clean tokens. The proposal latent never enters the persistent history. The current block is not regenerated after correction, so the effect of feedback starts from the next latent block.

This transaction also establishes an important train-inference consistency invariant. During training, the clean history fitted by the DiT consists of online-encoder latents of ground-truth tokens; at inference, after each block is completed, the next history is obtained from the final committed tokens through the same fixed-span encoder path. Hence, for the committed prefix,

$$\mathcal{K}_{\leq j}^Z = E_\phi(\mathcal{K}_{\leq j}^Y; \hat{\mathcal{S}}_{\leq j}), \quad (\text{G.36})$$

rather than $\mathcal{K}_{\leq j}^Z = Z_{\leq j}^{\text{prop}}$. The former keeps the continuous context of the DiT, the latent history of the condition stack, and the clean-token context of the discrete generator decoder pointing to the same finalized content. If the proposal were committed directly, any proposal-decode discrepancy would be permanently written into the latent cache and would cause cross-space context bias in subsequent blocks; fixed-span correction is designed precisely to eliminate this cumulative mismatch. Since the correction reuses the spans just predicted, it only aligns the representation content without reintroducing segmentation drift at inference.

Algorithm 1 summarizes the unified procedure.

G.6.1 Execution Details of MDM Prefix--Latent CFG

Condition preparation and scope. This path requires `decoder_cond_drop_scope=masked_only`, which guarantees that the prefix token KV contains no latent condition. We follow the branch subscripts a, b of

Algorithm 1 Unified block-wise inference of *Cola2*

Require: Prompt C , latent block size B , DiT steps S_{gen}

- 1: $z_{1:K_p}^p \leftarrow \mu(E_\phi(C))$
 - 2: Prefill the complete block-aligned prompt prefix into the persistent caches, and keep the remainder as the clean prefix of the first block
 - 3: **while** stopping criterion not met **do**
 - 4: Initialize the current latent block from Gaussian noise (and the optional clean prompt tail)
 - 5: **for** $i = 0, \dots, S_{\text{gen}} - 1$ **do**
 - 6: Update the DiT proposal on the committed corrected-latent prefix; restore the clean tail if present
 - 7: **end for**
 - 8: Predict span lengths from the proposal latents and construct the token-aligned latent condition
 - 9: Complete the current token block with the block-causal MDM decoder using the two-axis CFG in Eq. (3.37)
 - 10: Re-encode the final tokens with the fixed predicted spans to obtain Z_j^{corr}
 - 11: Atomically commit the finalized tokens, Z_j^{corr} , and the corresponding condition/encoder states
 - 12: **end while**
 - 13: **return** the generated token sequence
-

Appendix F.3. The refined condition is prepared only once per block:

$$H_j^Z = \text{Cond}\left(\text{Expand}(Z_j^{\text{prop}}, \hat{L}_{\mathcal{B}_j}); \mathcal{K}_{< j}^{\text{cond}}\right), \quad H_j^{Z, (b)} = bH_j^Z, \quad (\text{G.37})$$

where $\mathcal{K}_{< j}^{\text{cond}}$ is the committed condition-stack history and $\hat{L}_{\mathcal{B}_j}$ follows the existing duration rule. All four branches share the same H_j^Z ; $b = 0$ zeros the injected condition after refinement, without recomputing the duration or the condition stack.

Prefix-key masking and position indexing. Omitting the batch index, let $m_r \in \{0, 1\}$ indicate the validity of the r -th key slot, N_k the total number of key slots, and N_q the number of current query slots. The current block occupies the trailing index set $\mathcal{Q}_j = \{N_k - N_q + 1, \dots, N_k\}$. The two prefix visibilities correspond to

$$m_r^{(a)} = m_r(a + (1 - a)\mathbb{1}[r \in \mathcal{Q}_j]), \quad \text{pos}_r^{(a)} = \max\left(\sum_{s=1}^r m_s^{(a)} - 1, 0\right). \quad (\text{G.38})$$

For valid current-block queries, $m^{(a)}$ controls both key visibility and position indexing. With $a = 0$, only the valid keys of the current block are kept, and their positions are indexed contiguously from 0. If the first mixed block contains a clean prompt tail, the tail still belongs to $\mathcal{Q}_j$ and is not masked together with the persistent prefix.

Our implementation caches keys before RoPE and applies RoPE according to $\text{pos}^{(a)}$ after concatenating the cached and current keys. The two prefix branches thus reuse the same persistent KV without an additional cache or a modified RoPE operator, although the effective position indices differ between branches. Masking does not modify the tokens, the DiT proposal, or the condition-stack history. This rule adds no training objective; the first training block provides an input form without historical token keys, but we do not assume from this that all prefix-off inference states are in-distribution with respect to training.

Read-only branches, demasking, and commit. Let M_j be the number of refinement steps of the current block, $s \in \{0, \dots, M_j - 1\}$ the iteration index, and $\mathcal{M}_s$ the valid positions that are still masked. At each step, the branches with nonzero coefficients are computed on the same current token state, all with `update_kv=False`, and the logits are then combined according to Eq. (F.18). Denoting the combined logits at position t by $\ell_t^{(s)}$, the candidate token and confidence under greedy decoding are

$$\hat{y}_t^{(s)} = \arg \max_y \ell_t^{(s)}(y), \quad c_t^{(s)} = \log \text{softmax}(\ell_t^{(s)})(\hat{y}_t^{(s)}), \quad t \in \mathcal{M}_s. \quad (\text{G.39})$$

Under non-greedy decoding, candidates are sampled from the guided logits as configured. At each step, the highest-confidence candidates are kept according to the existing quota and the remaining positions stay masked; the known prompt tail does not participate in generation. Since $c_t^{(s)}$ is also derived from the guided logits, the two-axis weights affect both token selection and potentially the unmasking order, and confidence effects cannot be ruled out merely by the scale invariance of the greedy argmax.

After the block is finalized, the final tokens are written into the cache by a single latent-free clean forward pass (`update_kv=True`) without CFG, restoring full prefix visibility and normal position indexing. The original fixed-span feedback and corrected-history commit remain unchanged. The persistent token KV is determined only by finalized tokens; branch hidden states and masked states are temporary.

The two configuration parameters are `mdm_decoder_cfg_weight` ($\gamma_z - 1$) and `mdm_decoder_prefix_cfg_weight` ($\gamma_p - 1$). The prefix parameter defaults to 0, corresponding to $\gamma_p = 1$. For a fixed single-step state, the four branch logits can be cached and recombined with different weights; however, different weights change subsequent tokens, the unmasking order, and the committed history, so a complete rollout cannot share a single offline logits trajectory.

H Detailed Derivation of the Computational Cost

This section re-derives the computational cost according to the current training and inference execution graphs. There are two key differences: Stage 2 continues training the same compression encoder and discrete generator decoder and only additionally adds a detached DiT; and inference performs fixed-span feedback after the discrete tokens of each block are finalized, before committing the corrected latent history. Hence, the previous accounting of “Stage 1 VAE + Stage 2 DiT/decoder” or a one-way “DiT $\rightarrow$ decoder” no longer applies.

H.1 Notation

Below, one multiply-add counts as two FLOPs, and we count backbone linear layers, attention matrix multiplications, output heads, and auxiliary projections, while omitting element-wise operations such as embedding lookup, normalization, and softmax. Training includes the forward and backward passes, but excludes activation recomputation, optimizer updates, and communication. If these additional operations are counted, *Cola2* and AR should use the same convention.

For a module R , let N_R be the number of its backbone linear-layer parameters (excluding separate heads/interfaces), n_R its number of layers, d_R^q the total width of all query heads per layer, Q_R the cumulative number of processed positions, and $\mathcal{A}_R^{\text{exec}}$ the cumulative number of executed query-key pairs per layer. When all layers share the same shape,

$$\begin{aligned}\mathcal{C}_{R,\text{backbone}}^{\text{fwd}} &\approx 2N_R Q_R + 4n_R d_R^q \mathcal{A}_R^{\text{exec}}, \\ \mathcal{C}_{R,\text{backbone}}^{\text{train}} &\approx 6N_R Q_R + 12n_R d_R^q \mathcal{A}_R^{\text{exec}},\end{aligned}\tag{H.1}$$

where the attention term includes both $QK^\top$ and the attention-value product; heterogeneous layers are summed layer by layer. Let $\mathcal{A}_R^{\text{valid}}$ be the number of pairs allowed by the mask; it equals the executed amount only when the kernel actually skips invalid regions, and dense masks, tile boundaries, and padding must be charged according to the actual shapes.

If the vocabulary size is V , the output hidden width is d_{out} , and logits are actually computed at Q_{logit} positions, then

$$\mathcal{C}_{\text{head}}^{\text{fwd}} \approx 2V d_{\text{out}} Q_{\text{logit}}, \quad \mathcal{C}_{\text{head}}^{\text{train}} \approx 6V d_{\text{out}} Q_{\text{logit}}.\tag{H.2}$$

Weight sharing with the embedding does not eliminate this output projection, and supervising only a subset of positions does not automatically reduce the backbone Q_R . All of the above are approximations of the dominant matrix-multiplication terms; the total cost accumulates over actual calls rather than being counted by the number of distinct parameters.

A training sequence contains T valid tokens and, after compression,

$$K = \rho T.\tag{H.3}$$

At inference, let T_{new} be the number of newly generated tokens, K_{new} the number of corresponding latents, and $\rho_{\text{gen}} = K_{\text{new}}/T_{\text{new}}$ the realized ratio. Ignoring truncation at the beginning and end, the average number of tokens P covered by a latent block of size B and the number of blocks J satisfy

$$P \approx \frac{B}{\rho_{\text{gen}}}, \quad J \approx \frac{\rho_{\text{gen}} T_{\text{new}}}{B}. \quad (\text{H.4})$$

Duration prediction does not guarantee $\rho_{\text{gen}} = \rho^*$. The prompt tail in the first block, truncation in the last block, and batch padding may make the actual processed amount larger than the effective output, so we first give per-call counts and then approximations that ignore boundary effects.

H.2 Training: Persistent Representation/Decoder Path and an Additional Detached DiT

Stage 1 representation/decoder path. Denote the training cost of this path by $\mathcal{C}_{\text{rep}}(T, K)$. Clean, noisy, and drop are mutually exclusive condition states within the same main request, so the backbone forward pass is not multiplied by three; the additional requests of the generation-oriented $\mathcal{L}_{\text{gen}}$, however, must be counted separately. Below, $\mathcal{C}_E$ includes the encoder and the posterior projection, $\mathcal{C}_D^{\text{MDM}}$ includes the MDM token backbone, attention, and the corresponding output head, and the remaining operations such as conditioning and duration are listed as separate terms.

The MDM main request uses one clean encoder pass and one dual-layout decoder pass of length $2T$; $\mathcal{L}_{\text{encmask}}$ fixes the original spans and runs an additional masked-input encoder pass and a separate MDM request. Hence, when both requests are fully executed,

$$\begin{aligned} \mathcal{C}_{\text{rep}}^{\text{MDM}}(T, K) &\approx 2\mathcal{C}_E(T, K) + 2\mathcal{C}_D^{\text{MDM}}(2T) \\ &\quad + \mathcal{C}_{\text{cond/dur/KL}}^{\text{MDM}}, \end{aligned} \quad (\text{H.5})$$

where each single-pass cost is the average over the two requests. The two decoder requests together process about $4T$ backbone positions, with a dominant linear training term of $24N_D T$, where N_D is the number of MDM backbone parameters. Concatenating the two requests along the batch dimension does not change this count; the output head is charged by the actual logit positions, and repeated calls of the condition path are included in the last term. The two-axis inference CFG does not increase the number of training requests in either stage.

Let $\mathbb{E}_s$ denote the average over the actual training samples or batch layouts of Stage s , and T the corresponding number of valid tokens. Define

$$c_{\text{rep}}^{(s)} := \frac{\mathbb{E}_s[\mathcal{C}_{\text{rep}}(T, K)]}{\mathbb{E}_s[T]}, \quad s \in \{1, 2\}. \quad (\text{H.6})$$

This definition normalizes the actual executed amount by valid tokens. The representation path remains fully present in Stage 2, but when the sequence length, compression layout, or request configuration changes, $c_{\text{rep}}^{(1)} = c_{\text{rep}}^{(2)}$ need not hold.

Stage 2 detached DiT. The same clean encoder latent participates in the token objectives and also enters the DiT as $\bar{Z} = \text{sg}(Z)$. Suppose the training sequence has $J_{\text{tr}} = \lceil K/B \rceil$ latent blocks, and the last block has length $r_{\text{last}} = K - B(J_{\text{tr}} - 1) \in \{1, \dots, B\}$. Omitting the clean copy of the last block, $K_c = K - r_{\text{last}}$, and we define

$$\chi_G = \frac{K + K_c}{K}, \quad 1 \leq \chi_G < 2. \quad (\text{H.7})$$

When K is divisible by B , $r_{\text{last}} = B$ rather than 0. The dominant linear training term of the DiT backbone is

$$\mathcal{C}_{G, \text{linear}}^{\text{train}}(T, K) \approx 6N_G(K + K_c) = 6\chi_G \rho N_G T. \quad (\text{H.8})$$

Detachment only cuts the gradient to the encoder; it does not skip the backward pass of the DiT itself.

When $K = J_{\text{tr}}B$ and the clean stream is also computed block-causally, the numbers of valid pairs of clean and noisy queries correspond to the first $J_{\text{tr}} - 1$ blocks and all J_{tr} blocks, respectively:

$$\mathcal{A}_G^{\text{valid}} = B^2 \left(\sum_{j=1}^{J_{\text{tr}}-1} j + \sum_{j=1}^{J_{\text{tr}}} j \right) = K^2. \quad (\text{H.9})$$

Layouts with incomplete blocks remain $\mathcal{O}(K^2)$. The dominant attention training term is

$$\mathcal{C}_{G,\text{attn}}^{\text{train}} \approx 12n_G d_G^q \mathcal{A}_G^{\text{exec}}, \quad \mathcal{A}_G^{\text{valid}} = \mathcal{O}(\rho^2 T^2), \quad (\text{H.10})$$

where n_G and d_G^q follow the module-level notation. Let $\mathcal{C}_{G,\text{aux}}^{\text{train}}$ include the latent input/output and time/condition projections not counted in N_G . The average per-token cost of the added DiT is then

$$c_G^{(2)} = \frac{\mathbb{E}_2[\mathcal{C}_{G,\text{linear}}^{\text{train}} + \mathcal{C}_{G,\text{attn}}^{\text{train}} + \mathcal{C}_{G,\text{aux}}^{\text{train}}]}{\mathbb{E}_2[T]}. \quad (\text{H.11})$$

Hence ρ and ρ^2 describe the scaling of the linear length and the attention pairs under the same architecture and a comparable layout, rather than a fixed speedup factor for the complete system or an arbitrary kernel.

Total training cost and AR reference. If the two stages process D_1 and D_2 valid tokens, respectively, then

$$\mathcal{C}_{\text{Cola2}}^{\text{train}} \approx D_1 c_{\text{rep}}^{(1)} + D_2 (c_{\text{rep}}^{(2)} + c_G^{(2)}). \quad (\text{H.12})$$

With the same representation configuration, the first part can be merged into $(D_1 + D_2)c_{\text{rep}}$, but the DiT attention and auxiliary projections must still be retained.

Taking as reference an AR model with training budget D_{AR} and average per-token cost $c_{\text{AR}}^{\text{train}}$ under the same convention, define

$$r_{\text{rep}}^{(s)} = \frac{c_{\text{rep}}^{(s)}}{c_{\text{AR}}^{\text{train}}}, \quad r_G^{\text{train}} = \frac{c_G^{(2)}}{c_{\text{AR}}^{\text{train}}}. \quad (\text{H.13})$$

Then

$$\begin{aligned} R_{\text{train}} &:= \frac{\mathcal{C}_{\text{Cola2}}^{\text{train}}}{D_{\text{AR}} c_{\text{AR}}^{\text{train}}} \\ &\approx \frac{D_1}{D_{\text{AR}}} r_{\text{rep}}^{(1)} + \frac{D_2}{D_{\text{AR}}} (r_{\text{rep}}^{(2)} + r_G^{\text{train}}). \end{aligned} \quad (\text{H.14})$$

The AR cost $c_{\text{AR}}^{\text{train}}$ likewise includes attention and the output head. $6N_{\text{AR}}D_{\text{AR}}$ is only its backbone linear reference and cannot serve as the denominator for a complete cost that includes other operations. Compression mainly reduces the cost of the added prior and does not automatically make the entire training scale with ρ .

H.3 Inference: Proposal, Discrete Decoding, Feedback, and Commit

Excluding the one-time prompt encoding/prefill, the transaction of each generated block is

$$\text{DiT proposal} \rightarrow \text{DGD token decoding} \rightarrow \text{fixed-span re-encoding} \rightarrow \text{corrected-cache commit}. \quad (\text{H.15})$$

Subsequent attention still reads the prompt KV, so its history length cannot be dropped merely because prefill is excluded. Below, $\mathcal{C}_G^{\text{infer}}$ includes the DiT proposal/commit, $\mathcal{C}_D^{\text{infer}}$ includes condition preparation, token decoding, and the token-cache commit, and $\mathcal{C}_{\text{fb}}$ includes the feedback encoder and the corrected condition commit; the three parts do not overlap.

DiT proposal and corrected DiT commit. Let B_j^{exec} be the number of latent query slots actually processed in the j -th block, and $\nu_{G,j,i}$ the number of velocity branches actually executed at its i -th denoising step. Then

$$Q_G^{\text{prop}} = \sum_{j=1}^J B_j^{\text{exec}} \sum_{i=1}^{S_{\text{gen}}} \nu_{G,j,i} \quad (\text{H.16})$$

$$\approx \rho_{\text{gen}} \nu_{\text{gen}} S_{\text{gen}} T_{\text{new}}.$$

The second line assumes a uniform branch count ν_{gen} and ignores boundaries and padding. The proposal does not write to the persistent cache; corrected latents are committed only after feedback. To avoid folding attention into the linear coefficient, define

$$\kappa_G^{\text{commit}} := \frac{\mathcal{C}_{G,\text{linear}}^{\text{commit}}}{2N_G K_{\text{new}}}, \quad (\text{H.17})$$

where the numerator is only the backbone linear cost of the corrected DiT commit. If each new latent requires one additional full clean forward pass, $\kappa_G^{\text{commit}} \approx 1$ ignoring boundaries. The total cost is

$$\begin{aligned} \mathcal{C}_G^{\text{infer}} &\approx 2N_G Q_G^{\text{prop}} + \mathcal{C}_{G,\text{linear}}^{\text{commit}} \\ &\quad + \mathcal{C}_{G,\text{attn}}^{\text{infer}} + \mathcal{C}_{G,\text{aux}}^{\text{infer}}, \end{aligned} \quad (\text{H.18})$$

where the last two terms accumulate the attention and remaining input/output projections of the proposal and commit. If the current effective block width is b_j and the committed latent history length is P_j^Z , the numbers of valid pairs per branch with prefix on and off are $b_j(P_j^Z + b_j)$ and b_j^2 , respectively; these are accumulated over the actual denoising steps and commits, and the executed pair counts are substituted into Eq. (H.1).

MDM discrete generator decoder. Let W_j^{exec} be the query width actually processed in each refinement of the j -th block, W_j^{com} the number of positions processed in the final clean commit, M_j the number of refinement steps, and $\nu_{D,j,i}$ the number of decoder branches actually executed at step i . Then

$$\begin{aligned} \mathcal{C}_D^{\text{MDM}} &\approx 2N_D \sum_{j=1}^J \left(W_j^{\text{exec}} \sum_{i=1}^{M_j} \nu_{D,j,i} + W_j^{\text{com}} \right) \\ &\quad + \mathcal{C}_{D,\text{attn}}^{\text{MDM}} + \mathcal{C}_{\text{head/interface}}^{\text{MDM}}. \end{aligned} \quad (\text{H.19})$$

Ignoring the beginning, end, and padding, if ν_{dec} branches are computed at every step, the dominant linear term reduces to $2N_D \sum_j (\nu_{\text{dec}} M_j + 1) W_j$, where W_j is the effective block width; with a uniform $M_j = M$, it is approximately $2N_D (\nu_{\text{dec}} M + 1) T_{\text{new}}$.

The full two-axis CFG executes four branches, but $\nu_{D,j,i}$ counts the actual forward passes rather than only the nonzero coefficients. The cost is reduced only if the code skips zero-coefficient branches or merges identical branches before the call; multiplying logits by zero after the call saves no backbone FLOPs. The four branches share one proposal condition preparation per block, and only one clean-token commit without CFG is performed at the end.

To expand the effect of prefix visibility, let P_j^Y be the number of tokens committed before the j -th block (including the prompt), and $\nu_{j,i}^{\text{on}}$ the number of branches with the prefix kept at that step. With effective width W_j , the number of valid pairs over all refinements and the clean commit is

$$\begin{aligned} \mathcal{A}_D^{\text{valid}} &= \sum_{j=1}^J \left[\sum_{i=1}^{M_j} (\nu_{j,i}^{\text{on}} W_j P_j^Y + \nu_{D,j,i} W_j^2) \right. \\ &\quad \left. + W_j (P_j^Y + W_j) \right]. \end{aligned} \quad (\text{H.20})$$

The full two-axis CFG has $(\nu^{\text{on}}, \nu^{\text{dec}}) = (2, 4)$. Prefix-off branches only allow keys from the current block, but if the kernel does not skip the masked historical blocks, the actual FLOPs must use $\mathcal{A}_D^{\text{exec}}$:

$$\mathcal{C}_{D,\text{attn}}^{\text{MDM}} \approx 4n_D d_D^q \mathcal{A}_D^{\text{exec}}. \quad (\text{H.21})$$

$\mathcal{C}_{\text{head/interface}}^{\text{MDM}}$ includes all actual logit projections and one proposal duration/expansion/condition refinement. The recomputation of the corrected condition is attributed to feedback and is not double-counted here.

Fixed-span feedback and corrected condition commit. Each finalized block is re-encoded with fixed spans, without rerunning the dynamic segmenter. Let $N_{F,t}$ and $N_{F,z}$ be the numbers of linear-layer parameters of the span-local token pooling and the causal latent encoder, and $Q_{F,t}$ and $Q_{F,z}$ the numbers of positions they actually process. Then

$$\begin{aligned} \mathcal{C}_{\text{fb,enc}} &\approx 2N_{F,t}Q_{F,t} + 2N_{F,z}Q_{F,z} \\ &\quad + \mathcal{C}_{\text{fb,attn}} + \mathcal{C}_{\text{fb,proj}}, \end{aligned} \quad (\text{H.22})$$

where $\mathcal{C}_{\text{fb,proj}}$ includes projections such as the posterior head that are not counted in the above parameters; ignoring boundaries and padding, $Q_{F,t} \approx T_{\text{new}}$ and $Q_{F,z} \approx K_{\text{new}}$. A token-encoder depth of zero does not eliminate the computation of span-local pooling. Its valid attention pairs accumulate as $\sum_k L_k^2$ over spans, and the latent encoder is charged according to the causal prefix/current layout.

After obtaining Z_j^{corr} , the corrected condition-stack KV must also be rebuilt and committed; denoting its cumulative cost by $\mathcal{C}_{\text{cond,commit}}$, we have $\mathcal{C}_{\text{fb}} = \mathcal{C}_{\text{fb,enc}} + \mathcal{C}_{\text{cond,commit}}$. The proposal condition preparation and the corrected condition commit are two distinct computations, but neither is multiplied by four for the four CFG branches; the corrected DiT commit is already included in $\mathcal{C}_G^{\text{infer}}$.

Unified AR comparison convention. AR likewise excludes the one-time prefill. Let Q_{AR} be the number of positions actually processed by its incremental backbone, $\mathcal{A}_{\text{AR}}^{\text{exec}}$ the executed attention pairs, and $\mathcal{C}_{\text{head,AR}}^{\text{infer}}$ the output-projection cost. Then

$$\begin{aligned} \mathcal{C}_{\text{AR}}^{\text{infer}} &\approx 2N_{\text{AR}}Q_{\text{AR}} + 4n_{\text{AR}}d_{\text{AR}}^q \mathcal{A}_{\text{AR}}^{\text{exec}} \\ &\quad + \mathcal{C}_{\text{head,AR}}^{\text{infer}}. \end{aligned} \quad (\text{H.23})$$

If the initial KV length is T_p , processing Q_{AR} new queries consecutively yields $Q_{\text{AR}}T_p + Q_{\text{AR}}(Q_{\text{AR}} + 1)/2$ valid pairs; $Q_{\text{AR}} \approx T_{\text{new}}$ only ignores boundary differences in where logits are taken. Using denominators under the same convention, define

$$r_G^{\text{infer}} = \frac{\mathcal{C}_G^{\text{infer}}}{\mathcal{C}_{\text{AR}}^{\text{infer}}}, \quad r_{\text{DGD}} = \frac{\mathcal{C}_D^{\text{infer}}}{\mathcal{C}_{\text{AR}}^{\text{infer}}}, \quad r_{\text{fb}} = \frac{\mathcal{C}_{\text{fb}}}{\mathcal{C}_{\text{AR}}^{\text{infer}}}, \quad (\text{H.24})$$

so that

$$R_{\text{infer}} = \frac{\mathcal{C}_G^{\text{infer}} + \mathcal{C}_D^{\text{infer}} + \mathcal{C}_{\text{fb}}}{\mathcal{C}_{\text{AR}}^{\text{infer}}} = r_G^{\text{infer}} + r_{\text{DGD}} + r_{\text{fb}}. \quad (\text{H.25})$$

Using only the AR backbone linear term as reference and ignoring boundaries, the corresponding DiT contribution is

$$\frac{2N_G Q_G^{\text{prop}} + \mathcal{C}_{G,\text{linear}}^{\text{commit}}}{2N_{\text{AR}}T_{\text{new}}} \approx \rho_{\text{gen}}(\nu_{\text{gen}}S_{\text{gen}} + \kappa_G^{\text{commit}}) \frac{N_G}{N_{\text{AR}}}. \quad (\text{H.26})$$

This is not the total inference ratio. When end-to-end initialization is included, the numerator and denominator of Eq. (H.25) should add the prefill costs of *Cola2* and AR, respectively.

H.4 Serial Call Rounds, Latency, and Block Parallelism

We use $\mathcal{H}$ as a forward-round proxy, not as Transformer layer depth or actual latency. By convention, one module forward pass counts as one round, and independent CFG branches at the same step count as one group under ideal parallelism. Let $H_{D,j}$ be the number of call rounds for condition preparation, token decoding,

and clean-token commit of the j -th block, and $H_{F,j}$ the number of rounds for the feedback encoder and the corrected condition/DiT commit. Then

$$\mathcal{H}_{Cola2} \approx \sum_{j=1}^J (S_{\text{gen}} + H_{D,j} + H_{F,j}). \quad (\text{H.27})$$

Ignoring boundaries, AR corresponds to $\mathcal{H}_{\text{AR}} \approx T_{\text{new}}$. With H_D and H_F denoting the average per-block numbers of rounds,

$$\frac{\mathcal{H}_{Cola2}}{\mathcal{H}_{\text{AR}}} \approx \frac{\rho_{\text{gen}}}{B} (S_{\text{gen}} + H_D + H_F). \quad (\text{H.28})$$

Thus, the condition under which this idealized call-round proxy is smaller is

$$B > \rho_{\text{gen}} (S_{\text{gen}} + H_D + H_F), \quad (\text{H.29})$$

which is not a sufficient condition for end-to-end speedup. The decoding part of MDM contains about M_j refinement groups and one clean commit. If the CFG branches are invoked sequentially, every forward pass must be counted, and they cannot be treated as one round merely because the formula is parallelizable.

Let τ_{init} be the one-time initialization latency, $\tau_{G,j,i}$ the latency of the i -th DiT update of the j -th block (including the actual CFG scheduling), and $\tau_{\text{prep},j}$, $\tau_{D,j}$, and $\tau_{F,j}$ the latencies of condition preparation, token decoding/commit, and feedback/corrected-cache commit, respectively. Under the current serial transaction,

$$\tau_{Cola2} \approx \tau_{\text{init}} + \sum_{j=1}^J \left[\sum_{i=1}^{S_{\text{gen}}} \tau_{G,j,i} + \tau_{\text{prep},j} + \tau_{D,j} + \tau_{F,j} \right]. \quad (\text{H.30})$$

These latencies depend on module size, query/prefix length, branch batching, and hardware utilization, and should be measured on the corresponding execution paths; the same time cannot be assigned to every module forward pass.

H.5 KV Cache and Temporary States

Let K_{hist} and T_{hist} be the lengths of the committed latent/token histories, including the prompt. The DiT and feedback latent-encoder KV lie on the latent axis with length $\mathcal{O}(K_{\text{hist}})$; the condition-stack and token-decoder KV lie on the token axis with length $\mathcal{O}(T_{\text{hist}})$. The MDM token cache is bidirectional within a block and causal across blocks.

For $R \in \{G, E, C, D\}$, denoting the DiT, the feedback latent encoder, the condition stack, and the token decoder, respectively, let n_R be the number of KV layers and $d_R^{\text{KV}} = n_{\text{kv},R} d_{\text{head},R}$ the total KV-head width per layer, where $n_{\text{kv},R}$ and $d_{\text{head},R}$ are the number of KV heads and the per-head width. For an equal-length batch of N_{seq} sequences with s_{KV} bytes per cached element,

$$\begin{aligned} \text{Mem}_{\text{KV}} \approx 2s_{\text{KV}}N_{\text{seq}} & \left[(n_G d_G^{\text{KV}} + n_E d_E^{\text{KV}}) K_{\text{hist}} \right. \\ & \left. + (n_C d_C^{\text{KV}} + n_D d_D^{\text{KV}}) T_{\text{hist}} \right]. \end{aligned} \quad (\text{H.31})$$

The factor 2 accounts for keys and values; different lengths, KV layer shapes, or precisions are summed according to the actual allocation, and padding must also be counted. Under GQA, d_R^{KV} should be used instead of the hidden width.

The four MDM CFG branches share one persistent token KV, so Eq. (H.31) is not multiplied by four; temporary logits, branch hidden states, and attention workspaces still affect peak memory. Proposals and masked states do not accumulate as history, but the sizes of cache concatenation, RoPE, and attention temporaries may still depend on the prefix length. This formula therefore counts only persistent KV and does not equal the total peak memory.

In summary, compression shortens the sequence and the latent-axis history of the latent prior, while continuous representation learning, token decoding, CFG, and feedback all retain their own costs. End-to-end comparisons should use a unified FLOP accounting scope and separately report actually processed positions, serial call rounds, latency, and KV capacity, rather than equating any single metric with overall speedup.

I Final-Checkpoint Pretraining Recipe

This section describes the large-scale recipe used for the final checkpoint in Table 4. It is separate from the matched component and controlled scaling setup in Section 4.1.

I.1 Shared Training Configuration

The pretraining corpus uses OLMo 2 pretraining data retokenized with the Qwen3-4B tokenizer. The sequence length is 4096.

All stages use AdamW with $\beta = (0.9, 0.95)$ and weight decay 0.01. The EMA decay in Stages II and III is 0.9998. At stage boundaries, model parameters, optimizer state, the learning-rate scheduler, and the data cursor are restored.

I.2 Stage I: Dynamic-Compression VAE

Stage I processes approximately 52.43B source tokens. Dynamic segmentation compresses about two tokens into one latent on average, with maximum span length 8 and latent dimension 64. The decoder is initialized from Qwen3-4B-Base, and the VAE encoder embedding is copied from the same model; the remaining compression modules are initialized from scratch.

The Qwen-initialized parameters remain frozen for the first 5.24B source tokens and are subsequently updated at $0.1\times$ the main learning rate; the compression encoder also uses a $0.1\times$ multiplier. The learning rate is linearly warmed up from 10^{-6} to 10^{-4} over the first 5.24B source tokens and then cosine-decayed to 10^{-6} .

The decoder uses a causal dual-stream objective. Masked positions are supplied as [MASK] inputs, but the decoder self-attention remains causal rather than becoming bidirectional. Clean, noisy, and dropped latent conditions are sampled with proportions 0.4/0.3/0.3, and their three token-loss weights are all 1. The KL, duration, and encoder-mask auxiliary loss weights are 10^{-4} , 0.1, and 1, respectively, and the encoder-mask span ratio is 0.15.

I.3 Stage II: Joint VAE and Latent-DiT Training

Stage II restores the complete VAE checkpoint, optimizer, scheduler, and data cursor from Stage I. It introduces an approximately 4B-parameter latent DiT with 34 Transformer layers, hidden dimension 2560, 20 attention heads, and latent block size 8. This stage processes approximately 125.83B source tokens.

The VAE continues to use the Stage-I causal dual-stream objective, with the encoder-mask span ratio increased to 0.30. The DiT reads detached VAE latents, so its diffusion loss does not backpropagate into the VAE; the VAE is updated only by its compression and decoder objectives. The DiT uses velocity-prediction MSE, with timesteps sampled from LogitNormal(loc = 2, scale = 1).

The DiT learning rate is warmed up from 10^{-6} to 1.5×10^{-4} and then cosine-decayed. The schedule is defined over a nominal 2.1T-token horizon, although training terminates before reaching that horizon.

I.4 Stage III: Frozen VAE and Continued DiT Training

Stage III restores the complete DiT parameters, EMA, optimizer, learning-rate scheduler, and data cursor from the Stage-II checkpoint. The Stage-II EMA VAE encoder is frozen, and only the DiT continues to be optimized. The released model uses the matching EMA VAE decoder associated with this encoder.

The timestep distribution remains LogitNormal(loc = 2, scale = 1). This stage processes approximately 167.77B source tokens from the retokenized OLMo 2 corpus.

I.5 Source-Token Accounting

Table 5 Final-checkpoint pretraining budget. Token counts report source-token positions processed in each stage.

Stage	Source tokens	Primary update
Stage I	52.43B	Dynamic-compression VAE
Stage II	125.83B	VAE and latent DiT
Stage III	167.77B	Latent DiT only
Total	346.03B	—