COLA2: CO-ADAPTIVE HIERARCHICAL LANGUAGE MODELING WITH CONTINUOUS SEMANTIC COMPRESSION AND BALANCED DISCRETE GENERATION

Anonymous authors
Paper under double-blind review

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 validate the core designs, and on five downstream tasks with about 210B training tokens, Cola2 exhibits competitive scaling potential, providing empirical support for hierarchical language generation with continuous–discrete co-adaptation.

1 INTRODUCTION

Large language models (LLMs) have made remarkable progress under the autoregressive (AR) paradigm (Brown et al., 2020; Touvron et al., 2023), but the fixed left-to-right generation order imposes an inherent serial dependency on inference. Diffusion language models (DLMs) (Austin et al., 2021; Sahoo et al., 2024; Nie et al., 2025; Arriola et al., 2025) offer a new way to break this constraint. Cola DLM (Guo et al., 2026) 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 (Meshchaninov et al., 2025; Jiang et al., 2026; Liang et al., 2026). 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 (Uziel et al., 2026). 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

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

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 (Sahoo et al., 2024; Arriola et al., 2025) is jointly trained under clean, noisy, and dropped latent conditions, balancing semantic reconstruction with context-based autonomous generation, and classifier-free guidance (CFG) (Ho & Salimans, 2022) 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) (Peebles & Xie, 2023) learns the current latent distribution via Flow Matching (Lipman et al., 2023) 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.** We conduct component analyses of latent-space construction, the latent generator, and the decoder balance, and compare different generation paradigms on five downstream tasks with about 210B training tokens, demonstrating the highly competitive scaling potential of the Cola2 paradigm.

2 Related Work

Semantic compression and hierarchical language models. Fixed-granularity methods shorten the modeled sequence through chunking or downsampling (Nawrot et al., 2022; Yu et al., 2023). Adaptive compression methods dynamically aggregate bytes or tokens according to content (Pagnoni et al., 2025; Hwang et al., 2026; Qu et al., 2025). Concept-level methods organize generation through sentence representations or concept prediction (Barrault et al., 2024; Cao et al., 2026). In contrast, Cola2 controls the compression rate by negative feedback without an auxiliary rate loss, and combines reconstruction and generation supervision to learn compact, predictive latents.

Token-space language models. Autoregressive methods generate text serially, token by token (Brown et al., 2020; Grattafiori et al., 2024; Yang et al., 2025). Discrete diffusion methods predict multiple positions in parallel through iterative denoising (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024; Nie et al., 2026; Ye et al.; Sahoo et al., 2025), and have been further extended to block-level generation and token editing (Arriola et al., 2025; Bie et al., 2025; 2026). Continuous-augmented methods use paired continuous states to assist discrete denoising (Zheng et al., 2026; Zhou et al., 2025; Uziel et al., 2026). In contrast, Cola2 learns a semantic prior in a compressed latent space and realizes it with a discrete decoder.

Continuous and latent-space language models. Token-aligned methods perform diffusion or flow modeling on continuous representations (Li et al., 2022; Gong et al., 2022; Gulrajani & Hashimoto, 2023; Chen et al., 2026; Yang et al., 2026; Lee et al., 2026; Hu et al., 2026). Autoencoder-based latent-space methods separate latent generation from text decoding (Lovelace et al., 2023; Zhang et al., 2023; Meshchaninov et al., 2026a). Joint-learning methods further co-optimize the latent space and the generative model (Guo et al., 2026; Meshchaninov et al., 2026b). In contrast, Cola2 drives representation learning with discrete reconstruction and generation supervision while isolating the prior-fitting gradient.

3 COLA2: CONTINUOUS LATENT DIFFUSION FOR HIERARCHICAL TEXT MODELING

Cola2 generates text hierarchically: continuous latents carry compressed semantics, and a discrete decoder realizes them as tokens. Section 3.1 motivates the design and Section 3.2 presents the model; details are in Appendices A–H.

3.1 PROBABILISTIC AND INFORMATION-THEORETIC PERSPECTIVE

Let C be the observed context, Y the target text of T tokens, and $Z = (z_1, \dots, z_K)$, $z_k \in \mathbb{R}^{d_z}$, a continuous latent sequence with compression rate $\rho = K/T$. Cola2 factorizes

$$p(Y \mid C) = \int p_\theta(Z \mid C)\, p_\psi(Y \mid C, Z)\, \mathrm{d}Z. \tag{1}$$

With an encoder $q_\phi(Z \mid C, Y)$ as approximate posterior, the averaged negative conditional ELBO splits into a reconstruction term, the latent information $I(Y; Z \mid C)$, and $\mathbb{E}_C\, \mathrm{KL}(\bar{q}_\phi(Z \mid C) \,\|\, p_\theta(Z \mid C))$, where $\bar{q}_\phi$ is the aggregated posterior (Appendix A). Hence the encoder decides *what* enters the limited latent capacity, the decoder p_ψ realizes it, and the latent prior only needs to fit the distribution induced by the encoder. Given a high-level semantic variable S determined by Y, the target uncertainty decomposes as

$$\begin{aligned} H(Y \mid C) &= I(Y; Z \mid C) + H(Y \mid C, Z) && (2)\\ &= \underbrace{I(S; Z \mid C)}_{\text{semantic}} + \underbrace{I(Y; Z \mid S, C)}_{\text{local residual}} + \underbrace{H(Y \mid C, Z)}_{\text{discrete residual}}. && (3) \end{aligned}$$

This decomposition yields the three principles behind Cola2.

Semantic compression. Under $K = \rho T$, latents are worth modeling not merely because K is short, but because each position carries dense, predictive semantics: a high density $I(S; Z \mid C)/K$ and a large $I(Y^{\mathrm{tar}}; Z \mid C, Y^{\mathrm{vis}})$ about missing tokens (Appendix B). This motivates dynamic semantic spans and generation-oriented supervision.

Reconstruction–generation balance. For a degraded condition $\widetilde{Z}$ with $Y \to Z \to \widetilde{Z}$ given C, data processing gives $H(Y \mid C, Z) \le H(Y \mid C, \widetilde{Z})$: the weaker the latent condition, the more uncertainty the decoder must resolve on its own. As generated latents are imperfect, the decoder should be trained from reconstruction to autonomous generation (Appendix E).

Continuous–discrete co-adaptation. As $H(Y \mid C)$ is fixed by the data, the encoder determines the split in Eq. equation 2. Rather than fixing this split with a frozen encoder, Cola2 jointly minimizes $\mathbb{E}_{q_\phi}[-\log p_\psi(Y \mid C, Z)]$ over (ϕ, ψ) subject to $K/T \approx \rho^\star$, so that $\nabla_\phi \mathcal{L}_{\mathrm{tok}} \neq 0$ and the latent

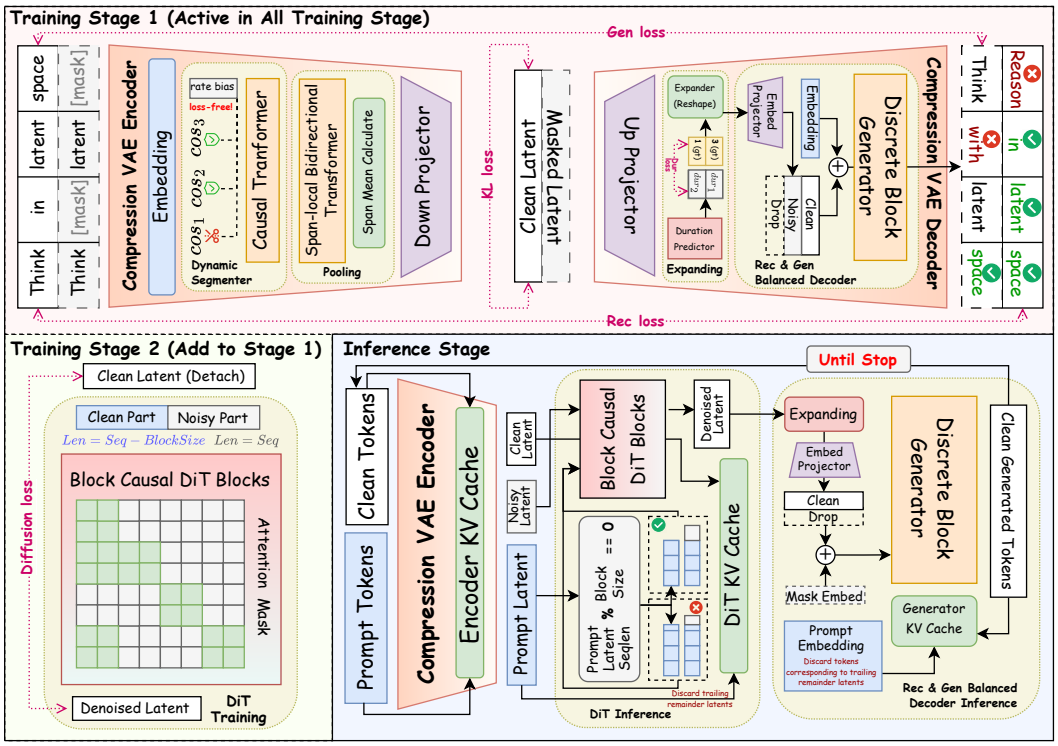


Figure 1: **Method overview.** Stage 1 jointly trains the compression encoder and the reconstruction–generation balanced decoder; Stage 2 additionally trains a DiT on detached latents; at inference, the DiT generates latents block by block and the decoder decodes them into tokens.

space adapts to the decoder that realizes it, while prior fitting is kept separate from representation learning (Appendix C).

3.2 Workflow

As shown in Figure 1, Cola2 instantiates these principles with three modules: a compression VAE encoder E_ϕ, a discrete generator decoder D_ψ, and a block-causal latent DiT G_θ:

$$(C, Y) \xrightarrow{E_\phi} Z, \qquad (C, Z) \xrightarrow{D_\psi} Y, \qquad \mathrm{sg}(Z) \xrightarrow{\mathcal{L}_{\mathrm{FM}}} G_\theta. \tag{4}$$

Architecture. E_ϕ dynamically partitions Y into K spans of length $1 \le L_k \le L_{\max}$, pools bidirectionally within each span, and outputs $q_\phi(z_k \mid C, Y) = \mathcal{N}(\mu_k, \mathrm{diag}(\sigma_k^2))$; a gradient-free controller holds ρ near $\rho^\star$ without an auxiliary loss (Appendix G.1). D_ψ predicts each span length $p_\psi(L_k \mid z_k)$, expands the latents into a token-aligned condition, and denoises the current token block with a block-causal masked-diffusion decoder (MDM; Appendix G.3). G_θ is causal across latent blocks of size B and bidirectional within them. High-level generation thus iterates over only $K \approx \rho T$ positions.

Stage 1: co-adaptive compression and balanced decoding. To cover the spectrum from reconstruction to generation, each latent block samples a condition-reliability state $A \in \{\mathsf{cond}, \mathsf{noise}, \mathsf{drop}\}$:

$$\widetilde{Z}_{\mathsf{cond}} = Z, \qquad \widetilde{Z}_{\mathsf{noise}} = (1 - u^{\mathrm{c}})Z + u^{\mathrm{c}}\varepsilon, \qquad \widetilde{Z}_{\mathsf{drop}} = \varnothing, \qquad \varepsilon \sim \mathcal{N}(0, I), \tag{5}$$

where $\varnothing$ removes only the current latent condition while keeping the token context. Denoting by $\mathcal{L}_{\mathrm{tok}}^A$ the MDM loss under state A, the balanced loss is $\mathcal{L}_{\mathrm{bal}} = \sum_A \lambda_A \mathcal{L}_{\mathrm{tok}}^A$. A generation-oriented

loss $\mathcal{L}_{\text{gen}}$ masks part of the encoder input under a fixed span layout and supervises the resulting latents through token prediction. The posterior is regularized by a dimension-normalized KL,

$$\mathcal{L}_{\text{KL}} = \frac{K_{\text{ref}} d_{\text{ref}}}{T_{\text{ref}}} \frac{1}{K d_z} \sum_{k=1}^{K} \text{KL}(q_\phi(z_k \mid C, Y) \,\|\, \mathcal{N}(0, I)), \tag{6}$$

and the duration head is supervised by the true span lengths ($\mathcal{L}_{\text{dur}}$), giving $\mathcal{L}^{(1)} = \mathcal{L}_{\text{bal}} + \lambda_{\text{gen}}\mathcal{L}_{\text{gen}} + \beta_{\text{KL}}\mathcal{L}_{\text{KL}} + \lambda_{\text{dur}}\mathcal{L}_{\text{dur}}$. All token losses back-propagate into E_ϕ through Z, so the encoder and decoder jointly determine the information allocation in Eq. equation 2.

Stage 2: detached latent diffusion. Stage 2 keeps $\mathcal{L}^{(1)}$ and adds latent-prior fitting on $\bar{Z} = \text{sg}(Z)$. Each latent block j shares a logit-normal timestep u_j, and with the linear Flow Matching path

$$z_u = (1-u)\bar{z} + u\epsilon, \qquad \mathcal{L}_{\text{FM}} = \mathbb{E}\Big[\big\|G_\theta(z_u, u; \bar{Z}_{<j}) - (\epsilon - \bar{z})\big\|_2^2\Big], \qquad \epsilon \sim \mathcal{N}(0, I), \tag{7}$$

the total objective is $\mathcal{L}^{(2)} = \mathcal{L}^{(1)} + \lambda_{\text{FM}}\mathcal{L}_{\text{FM}}$ with $\nabla_\phi \mathcal{L}_{\text{FM}} = 0$ and $\nabla_\phi \mathcal{L}^{(1)} \neq 0$. The prior thus cannot pull the latents toward easy-to-fit but less informative solutions, while representation and realization stay co-adapted (Appendices C, G.5).

Inference: propose–decode–re-encode–commit. The prompt is encoded with the posterior mode, and only its block-aligned prefix is committed to the cache (Appendix G.6). Each generated block j then follows

$$Z_j^{\text{prop}} \xrightarrow{D_\psi} Y_j \xrightarrow{E_\phi(\,\cdot\,;\widehat{\mathcal{S}}_j)} Z_j^{\text{corr}} \xrightarrow{\text{commit}} \mathcal{K}_{j+1}, \tag{8}$$

where the proposal Z_j^{prop} only decodes the current tokens Y_j, which are re-encoded under the predicted span layout $\widehat{\mathcal{S}}_j$, and only Z_j^{corr} enters the history. The latent history is thus produced by the same encoder path as in training and stays aligned with the generated tokens, preventing continuous–discrete context drift.

Two-level CFG. The DiT applies classifier-free guidance (CFG) to the historical latent condition in velocity space,

$$\hat{v}_\gamma = v_{\text{u}} + \gamma_{\text{gen}}(v_{\text{c}} - v_{\text{u}}), \tag{9}$$

and integrates $\hat{v}_\gamma$ with S_{gen} Euler steps. The decoder applies prefix–latent two-axis CFG in logit space: with ℓ_{ab} the logits when the token prefix (a) and current latent condition (b) are kept (1) or masked (0),

$$\ell_{\gamma_{\text{p}},\gamma_{\text{z}}} = \sum_{a,b\in\{0,1\}} w_a(\gamma_{\text{p}})\, w_b(\gamma_{\text{z}})\, \ell_{ab}, \qquad w_1(\gamma) = \gamma, \quad w_0(\gamma) = 1 - \gamma, \tag{10}$$

so γ_{z} trades off semantic reconstruction against autonomous generation and γ_{p} controls reliance on the token history (Appendices F, F.3). On $K = \rho T$ positions, the DiT's linear and attention costs shrink to $\mathcal{O}(\rho)$ and $\mathcal{O}(\rho^2)$ of the token axis (Appendix H).

4 Experiments

4.1 Experimental Setup

We train on the OLMo 2 corpus (OLMo et al., 2024). Component analyses report LAMBADA (Paperno et al., 2016) accuracy; in Sections 4.2–4.3, CFG fixes the decoder in latent-conditioned reconstruction mode. Scaling experiments train on about 210B tokens and report exact-match accuracy on RACE (Lai et al., 2017), OBQA (Mihaylov et al., 2018), Story Cloze (Mostafazadeh et al., 2016), LAMBADA, and SQuAD (Rajpurkar et al., 2016). We compare with AR (Brown et al., 2020), LLaDA (Nie et al., 2026), Cola DLM (Guo et al., 2026), and Cola-MDM, an uncompressed variant ($\rho = 1$) that shares the decoder of Cola2-MDM and thus isolates the effect of compression; for rate control, we compare with the auxiliary rate loss of the dynamic compression method DLCM (Qu et al., 2025); all models have about 2.4B parameters and use identical data and evaluation. By default, a latent covers 2 tokens on average, $d_z = 64$, $L_{\max} = 8$, and inference uses 8 DiT and 4 demasking steps. Full settings are given in Appendix I.

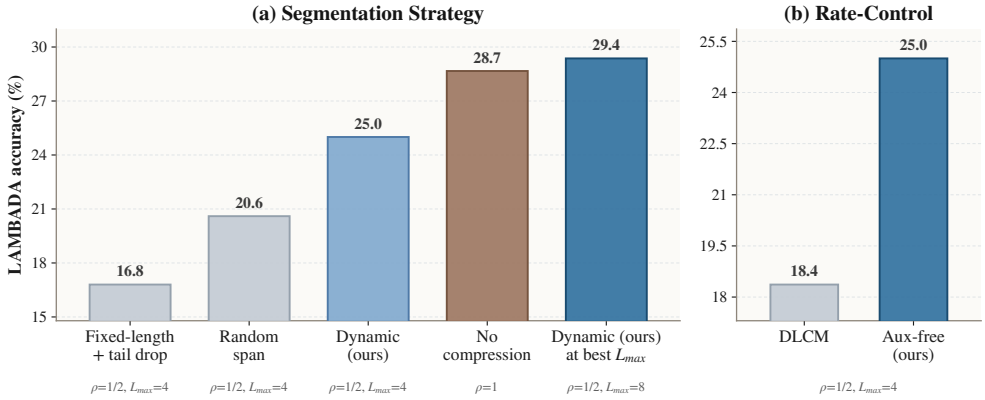


Figure 2: **Segmentation and rate control.** Left: segmentation strategy. Right: rate control.

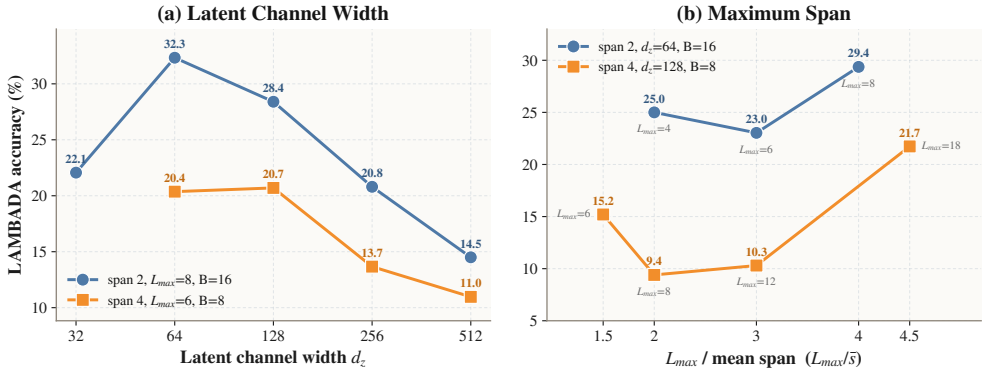


Figure 3: **Latent capacity.** Left: latent width d_z. Right: maximum span length $L_{\max}$.

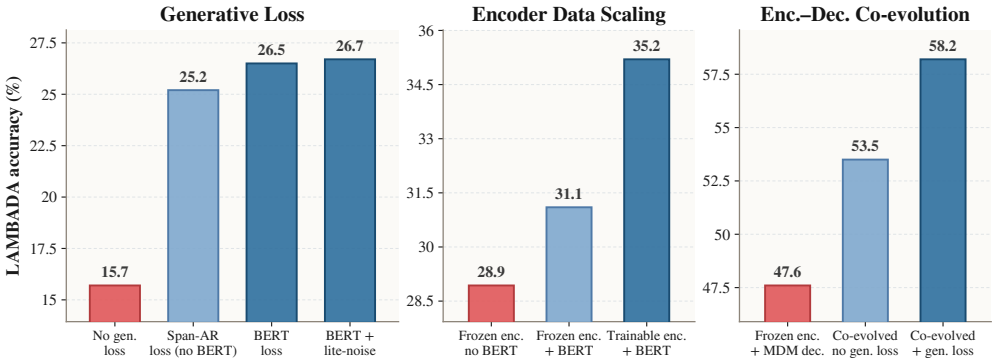


Figure 4: **Generative supervision.** Left: objective. Middle: frozen vs. trainable encoder. Right: co-evolution.

4.2 Building the Semantic Latent Space

Lemma 4.1 (Generative Supervision Promotes Predictive Latents)**.** *If the supervised tokens are hidden from a trainable encoder, reducing the generative objective drives the latent to retain information predictive of the missing tokens; with a frozen encoder, it only improves how the decoder uses existing latent information (Appendix B).*

Obs. 1 Dynamic segmentation is essential. With $\rho = 1/2$ and $L_{\max} = 4$, it beats random and fixed-length spans by 4.4 and 8.2 points; with $L_{\max} = 8$ it reaches 29.4%, surpassing the uncompressed model (28.7%; Figure 2). **Obs. 2 Feedback rate control beats an auxiliary rate loss** (25.0% vs. 18.4% for DLCM), as it avoids competition between the rate loss and repre-

Table 1: **Best setting per span length.**

Span len.	d_z	$L_{\max}$	LAMBADA (%)
2	64	8	**32.3**
3	128	8	26.9
4	256	18	25.2

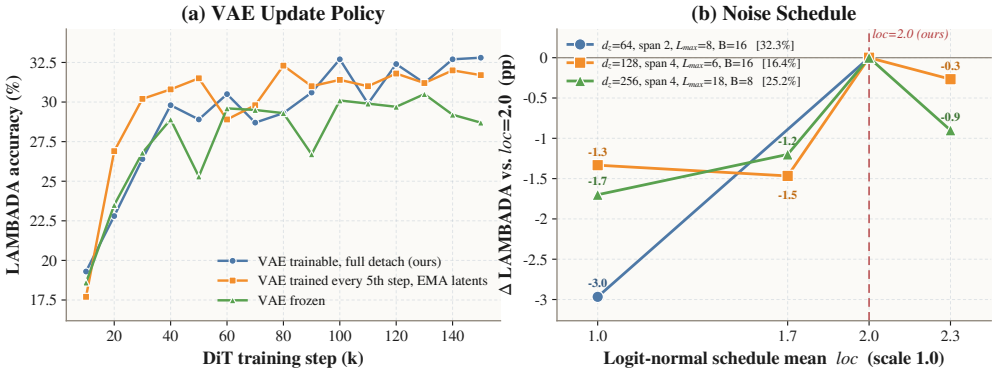


Figure 5: **Stage-2 training.** Left: VAE update policy. Right: accuracy change relative to $\mathrm{loc} = 2.0$.

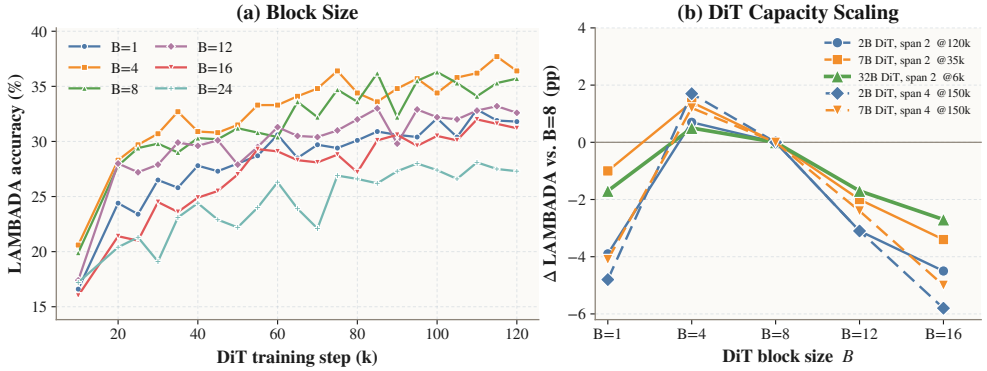


Figure 6: **DiT block size.** Left: block-size sweep. Right: sensitivity across DiT sizes.

sentation learning. **Obs. 3 Encoder layout matters more than size.** A four-layer 0+2+2 encoder (29.6%) beats all six-layer variants ($\leq$ 26.3%), and intra-span bidirectional pooling is the best aggregator (Figure 11). **Obs. 4 Wider latents are not always better.** Accuracy peaks at $d_z = 64$ (32.3%) and falls to 14.5% at $d_z = 512$ (Figure 3, left). **Obs. 5 Compression rate, d_z, and $L_{\max}$ must be tuned jointly.** Even at the best setting, accuracy falls from 32.3% to 25.2% as the span length grows from 2 to 4 (Table 1), and the effect of $L_{\max}$ depends on truncation (Figure 3, right). **Obs. 6 Generative supervision is key; its form is secondary.** Pure reconstruction reaches only 15.7%, whereas Span-AR and BERT-style objectives reach 25.2% and 26.5%. The BERT-style gain is 2.2 points with a frozen encoder (28.9% $\rightarrow$ 31.1%) and grows to 35.2% with a trainable one (Figure 4), as predicted by Lemma 4.1. **Obs. 7 Encoder and decoder should co-evolve.** With the MDM decoder, end-to-end co-evolution lifts a frozen-encoder baseline from 47.6% to 53.5%, and the generative loss further to 58.2% (Figure 4, right).

4.3 Learning the Latent Generator

Lemma 4.2 (Prior Fitting and Representation Learning Should Be Decoupled)**.** *The Flow Matching gradient only makes latents easier to fit and need not align with the token objective; stop-gradient keeps the prior from rewriting the encoder while retaining decoder-driven updates (Appendix C).*

Table 2: **DiT gradient into the encoder.**

Gradient policy	20k	80k	140k
Full detachment (ours)	21.4	**30.4**	**33.1**
Injection every 10 steps	**23.6**	29.9	32.1
Diff. (ours − injection)	−2.2	+0.5	+1.0

Obs. 8 Keep training the VAE, but detach the DiT gradient. A trainable VAE reaches 32.8% vs. 28.7% when frozen (Figure 5, left). Injecting the DiT gradient leads by 2.2 points at 20k steps but trails by 1.0 point at 140k (Table 2): as Lemma 4.2 predicts, the prior rewrites latents to be easy to fit rather than informative. **Obs. 9 The noise schedule must match the representation.** All configurations peak at $\mathrm{loc} = 2.0$, but deviations cost 0.3–3.0 points depending on d_z and compression (Figure 5, right), since VAE and diffusion noise jointly set the effective SNR (Lemma D.1). **Obs. 10 Moderate blocks are best, and larger DiTs tolerate larger blocks.** For a 2B DiT, $B = 4$ gives 36.4% vs. 31.8% ($B = 1$) and

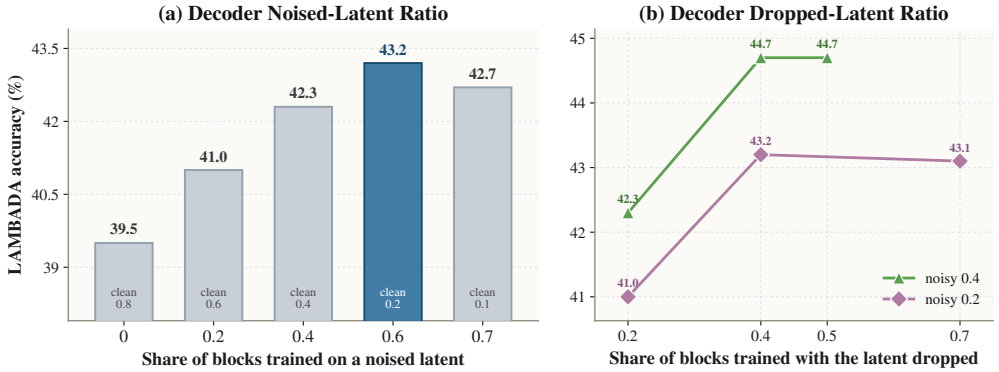


Figure 7: **Decoder data partition.** Left: noisy/clean ratio. Right: drop ratio.

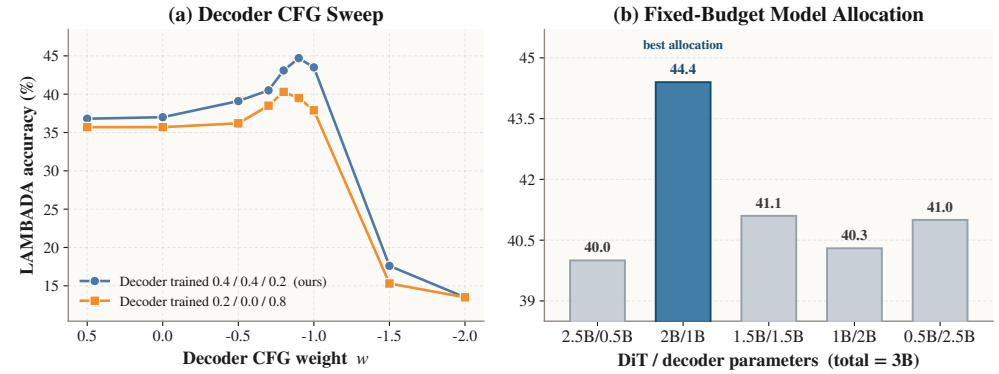


Figure 8: **Decoder CFG and allocation.** Left: CFG weight. Right: DiT/decoder split (3B total).

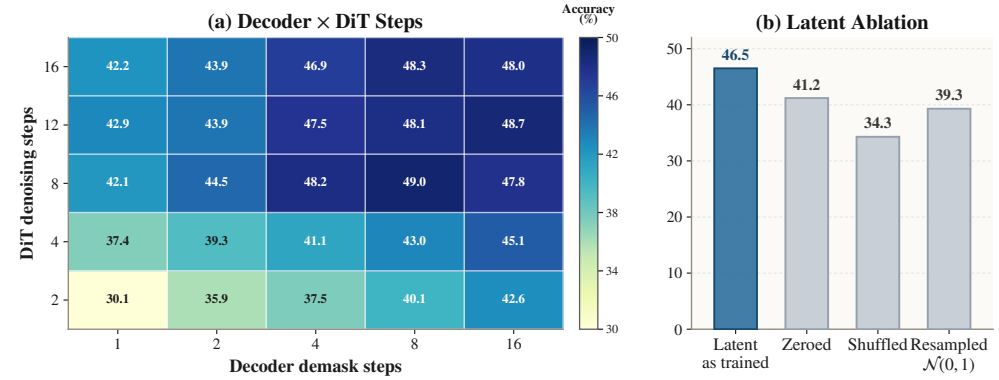


Figure 9: **Decoder inference and latent use.** Left: DiT/demasking steps. Right: latent corruption.

27.3% ($B = 24$); the drop from $B = 8$ to 16 shrinks from 4.5 (2B) to 3.4 (7B) and 2.7 points (32B; Figure 6), as a larger DiT can jointly model more latents per block. **Obs. 11 The generator needs strong CFG and about 8 steps.** Guidance lifts accuracy from 12.2% to 37.1%, and gains saturate at 8 steps (Appendix I).

4.4 Balancing the Reconstruction–Generation Decoder

Lemma 4.3 (Latent-Condition Strength Determines Residual Generation)**.** *A more reliable latent condition leaves less residual uncertainty for the decoder to generate on its own, so mixed* cond/noise/drop *training spans the continuum from reconstruction to autonomous generation (Appendix E).*

Obs. 12 Decoder factorization matters little. AR-MTP and MDM decoders differ by at most 1.4 points with alternating leads (Appendix I); we use MDM. **Obs. 13 Train mostly on noisy and**

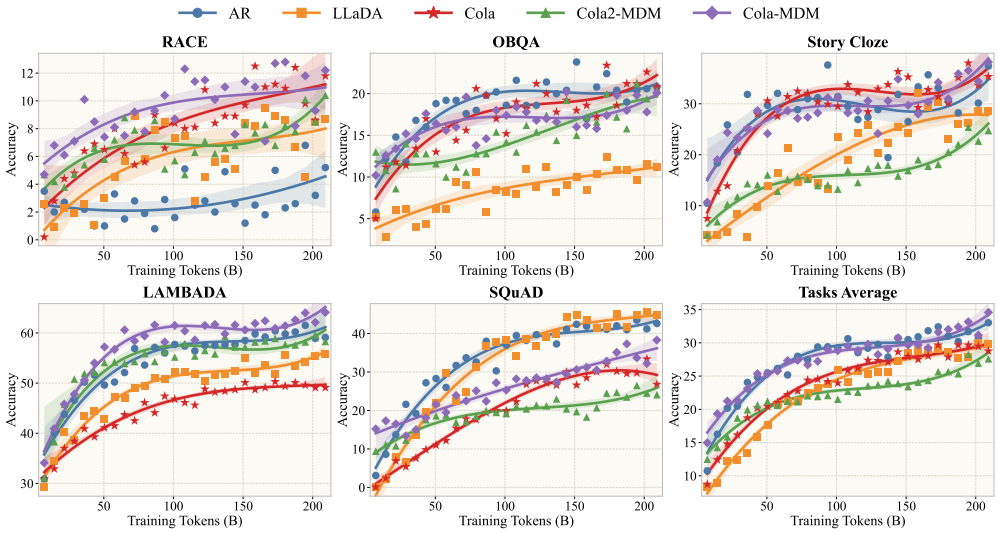


Figure 10: **Scaling with training tokens** on five downstream tasks and their average.

dropped conditions, but keep clean ones. With drop fixed at 0.2, moving noisy/clean from $0.0/0.8$ to $0.6/0.2$ lifts accuracy from 39.5% to 43.2%, and raising drop to 0.4 adds 2.2–2.4 points before saturating (Figure 7). We adopt $0.4/0.4/0.2$ (drop/noisy/clean; 44.7%), in line with Lemma 4.3. **Obs. 14 Decoder CFG must match the training mixture.** The balanced decoder peaks at $w_{\text{dec}} = -0.9$ (44.7%), whereas a clean-heavy decoder reaches at most 40.3% (Figure 8, left): guidance cannot compensate for imbalanced training. **Obs. 15 Under a fixed budget, the DiT matters more.** Although a larger decoder helps, with 3B in total a 2B DiT plus a 1B decoder (44.4%) beats other splits by 3.3–4.4 points (Figure 8, right). **Obs. 16 Demasking saturates at about 8 steps.** 8 DiT/8 demasking steps are best (49.0%), and more demasking partly offsets too few DiT steps; the default $8/4$ is only 0.8 points lower (Figure 9, left). **Obs. 17 The decoder relies on aligned latents.** Zeroing, resampling, and shuffling the latent lower accuracy from 46.5% to 41.2%, 39.3%, and 34.3% (Figure 9, right). Realistic but misaligned shuffled latents hurt most, confirming that the decoder follows latent content.

4.5 Scaling Experiments

Obs. 18 The MDM decoder yields the strongest scaling. Cola-MDM reaches a task average of about 34.5%, above AR (about 33%), and about 64% on LAMBADA, 15 points above Cola DLM (Figure 10); the weakness of Cola DLM thus stems from its decoder rather than the hierarchy. **Obs. 19 Compression preserves semantic tasks, with costs on exact-token tasks.** Cola2-MDM matches AR on LAMBADA (about 61%) and leads on RACE (10.5% vs. 4.5%), trailing mainly on SQuAD and Story Cloze, which require verbatim reproduction. **Obs. 20 The compressed variant converges more slowly but keeps closing the gap.** As each latent summarizes more tokens, early progress is slower; after about 150B tokens, the average gap to LLaDA shrinks from about 4 to 1.5 points without saturating, and the selected decoder CFG remains near-optimal (Appendix I).

5 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 uses context to handle residual uncertainty. Token-prediction gradients drive encoder–decoder co-adaptation, while latent-prior fitting remains decoupled from representation learning. Systematic component analyses and evaluations on five downstream tasks, with training scaled to approximately 210B tokens, support these design principles and demonstrate competitive scaling potential for hierarchical language generation.

AI USE STATEMENT

In this work, we used generative AI tools (e.g., large language models) solely for language polishing and grammar checking to improve the readability of the manuscript. We have not used generative AI tools for research ideation, data generation, code writing, theoretical derivation, or literature review. We have carefully reviewed all AI-assisted work. Specifically, all AI-polished text was checked word-for-word by all authors to ensure that it accurately reflects our original scientific meaning and intent. We take full responsibility for the final content of this work, including all text and claims.

ETHICS STATEMENT

This work primarily involves foundational algorithmic and theoretical research in machine learning and artificial intelligence. Our study does not involve human subjects, the collection of sensitive personal data, or any practices that violate research ethics guidelines. All datasets used in our experiments are publicly available, open-source benchmark datasets. We are committed to upholding open science and research transparency. While our proposed methodology could theoretically be applied to a wide range of downstream applications, we do not foresee any direct negative social impacts, nor do we anticipate concerns regarding discrimination, bias, or fairness. There are no potential conflicts of interest in this research.

REFERENCES

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, pp. 50726–50753, 2025.

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.

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.

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.

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.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

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.

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.

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.

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.

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.

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.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. URL https://arxiv.org/abs/2207.12598.

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.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. In *International Conference on Learning Representations*, volume 2026, pp. 149273–149313, 2026.

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. URL https://arxiv.org/abs/2605.07748.

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*, pp. 785–794, 2017.

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.

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.

Jiajun Liang, Yucheng Liao, Yukang Cao, Jiazhe Wei, Ken Li, Wende Tan, Jiankun Zhang, ZY Cui, Jingkang 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. URL https://arxiv.org/abs/2608.02602.

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. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q. Weinberger. Latent diffusion for language generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 56998–57025, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b2a2bd5d5051ff6af52e1ef60aefd255-Abstract-Conference.html.

Viacheslav Meshchaninov, Egor Chimbulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry P. Vetrov. COSMOS: Compressed and smooth latent space for text diffusion modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1506930fb75c82246e4d8648a66e4b27-Abstract-Conference.html.

Viacheslav Meshchaninov, Egor Chimbulatov, 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, 2026a.

Viacheslav Meshchaninov, Alexander Shabalin, Egor Chimbulatov, 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*, 2026b.

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*, pp. 2381–2391, 2018.

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*, pp. 839–849, 2016.

Piotr Nawrot, Szymon Tworkowski, 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*, pp. 1559–1571, 2022.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

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.

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.

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)*, pp. 9238–9258, 2025.

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)*, pp. 1525–1534, 2016.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Peebles_Scalable_Diffusion_Models_with_Transformers_ICCV_2023_paper.html.

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.

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*, pp. 2383–2392, 2016.

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.

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.

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. URL https://arxiv.org/abs/2302.13971.

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. URL https://arxiv.org/abs/2603.20210.

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.

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.

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*.

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.

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.

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, pp. 34212–34242, 2026.

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.

A Detailed Derivations of the Probabilistic Model and Information Decomposition

A.1 Conditional ELBO

During training, the encoder $q_\phi(Z \mid C, Y)$ serves as the approximate posterior, where ϕ denotes the encoder parameters. By Jensen's inequality, the conditional log-likelihood in Eq. equation 1 satisfies

$$\log p(Y \mid C) \geq \mathbb{E}_{q_\phi(Z|C,Y)} \left[\log p_\psi(Y \mid C, Z) - \log \frac{q_\phi(Z \mid C, Y)}{p_\theta(Z \mid C)} \right]. \tag{11}$$

Define the conditional aggregated posterior as

$$\bar{q}_\phi(Z \mid C) = \int q_\phi(Z \mid C, Y) p_{\text{data}}(Y \mid C) \, \mathrm{d}Y. \tag{12}$$

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

$$\begin{aligned} &\mathbb{E} \, \mathrm{KL}(q_\phi(Z \mid C, Y) \| p_\theta(Z \mid C)) \\ &= I(Y; Z \mid C) + \mathbb{E}_C \, \mathrm{KL}(\bar{q}_\phi(Z \mid C) \| p_\theta(Z \mid C)) \, . \end{aligned} \tag{13}$$

Eq. equation 13 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 $p_\theta(Z \mid C)$, not part of the definition of the generative model itself.

A.2 Information Identities and Semantic Decomposition

Conditional mutual information is defined as

$$I(Y; Z \mid C) = H(Y \mid C) - H(Y \mid C, Z), \tag{14}$$

and rearranging directly yields Eq. equation 2 in the main text.

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

$$\begin{aligned} I(Y; Z \mid C) &= I(S, Y; Z \mid C) \\ &= I(S; Z \mid C) + I(Y; Z \mid S, C), \end{aligned} \tag{15}$$

which yields the semantic decomposition in Eq. equation 3. Under the compression constraint $K = \rho T$, we further define the semantic information density per latent position as

$$\mathcal{D}_{\text{sem}} = \frac{I(S; Z \mid C)}{K}. \tag{16}$$

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

$$\begin{aligned} I(Y^{\text{tar}}; Z \mid C, Y^{\text{vis}}) = H(Y^{\text{tar}} \mid C, Y^{\text{vis}}) \\ - H(Y^{\text{tar}} \mid C, Y^{\text{vis}}, Z). \end{aligned} \tag{17}$$

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

$$\begin{aligned} &\mathbb{E}_{C,Y}\, \mathrm{KL}(q_\phi(Z \mid C, Y) \| p_0(Z \mid C)) \\ &= I(Y; Z \mid C) + \mathbb{E}_C\, \mathrm{KL}(\bar{q}_\phi(Z \mid C) \| p_0(Z \mid C)) \\ &\geq I(Y; Z \mid C). \end{aligned} \tag{18}$$

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 $K d_z$. Since $K = \rho T$, the information capacity available per token follows the approximate scaling

$$\frac{1}{T} I(Y; Z \mid C) \lesssim \rho d_z \bar{r}. \tag{19}$$

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

$$Y_M \longrightarrow (C, Y_{\bar{M}}) \longrightarrow Z^M. \tag{20}$$

By the data processing inequality,

$$I(Y_M; Z^M \mid C, M) \le I(Y_M; Y_{\bar{M}} \mid C, M). \tag{21}$$

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 $p_\eta(Y_M \mid C, Z^M, M)$, the cross-entropy can be written as

$$\begin{aligned} \mathcal{L}_{\text{gen}}^{\text{mask}} &= H(Y_M \mid C, Z^M, M) \\ &\quad + \mathbb{E}\,\mathrm{KL}\big(q(Y_M \mid C, Z^M, M) \| p_\eta(Y_M \mid C, Z^M, M)\big)\,. \end{aligned} \tag{22}$$

If the predictor is conditionally optimal, then

$$\begin{aligned} (\mathcal{L}_{\text{gen}}^{\text{mask}})^\star &= H(Y_M \mid C, Z^M, M) \\ &= H(Y_M \mid C, M) - I(Y_M; Z^M \mid C, M). \end{aligned} \tag{23}$$

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 Cola DLM (Guo et al., 2026).

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

$$\mathcal{L}_{\text{rep}}(\phi, \psi) = \mathcal{L}_{\text{bal}} + \lambda_{\text{gen}}\mathcal{L}_{\text{gen}} + \lambda_{\text{dur}}\mathcal{L}_{\text{dur}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}, \tag{24}$$

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

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{rep}} + \lambda_{\text{FM}}\mathcal{L}_{\text{FM}}. \tag{25}$$

Define

$$g_{\text{rep}} = \nabla_\phi \mathcal{L}_{\text{rep}}, \qquad g_{\text{FM}} = \nabla_\phi \mathcal{L}_{\text{FM}}. \tag{26}$$

A single SGD update is

$$\phi^+ = \phi - \eta\big(g_{\text{rep}} + \lambda_{\text{FM}} g_{\text{FM}}\big). \tag{27}$$

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

$$\Delta\mathcal{L}_{\text{rep}} \approx -\eta\|g_{\text{rep}}\|_2^2 - \eta\lambda_{\text{FM}}\langle g_{\text{rep}}, g_{\text{FM}}\rangle. \tag{28}$$

Therefore, whenever

$$\langle g_{\text{rep}}, g_{\text{FM}}\rangle < -\frac{\|g_{\text{rep}}\|_2^2}{\lambda_{\text{FM}}}, \tag{29}$$

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,

$$z_0 = g_\phi(x, \epsilon_0), \qquad \epsilon_0 \sim \mathcal{N}(0, I). \tag{30}$$

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

$$z_t = \alpha_t z_0 + \sigma_t \epsilon, \qquad v_t^\star = a_t z_0 + b_t \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \tag{31}$$

and we define

$$r = v_\theta(z_t, t, c) - v_t^\star, \qquad \ell_v = \|r\|_2^2, \tag{32}$$

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

$$\nabla_{z_0}\ell_v = 2\big(\alpha_t J_{v,z_t}^\top - a_t I\big)r, \tag{33}$$

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:

$$2J_{c,z_0}^\top J_{v,c}^\top r. \tag{34}$$

Finally,

$$\nabla_\phi \ell_v = (\nabla_{z_0}\ell_v)^\top \frac{\partial z_0}{\partial \phi}. \tag{35}$$

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

$$\begin{aligned}\mathbb{E}\|v(W) - u_t\|_2^2 = \mathbb{E}\|v(W) - \mathbb{E}[u_t \mid W]\|_2^2 \\ + \mathbb{E}\|u_t - \mathbb{E}[u_t \mid W]\|_2^2.\end{aligned} \tag{36}$$

Hence

$$v^\star(W) = \mathbb{E}[u_t \mid W], \qquad \mathcal{L}_{\text{FM}}^\star = \mathbb{E}[\text{Var}(u_t \mid z_t, t, c)]\,. \tag{37}$$

Eq. equation 37 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

$$q_{\text{sem}}(z \mid S) = \mathcal{N}(mSe_1, I), \quad m > 0, \qquad q_{\text{col}}(z \mid S) = \mathcal{N}(0, I). \tag{38}$$

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

$$\mathcal{L}_{\text{FM}}(q_{\text{col}}) < \mathcal{L}_{\text{FM}}(q_{\text{sem}}), \qquad I(S; Z_{\text{col}}) < I(S; Z_{\text{sem}}). \tag{39}$$

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

$$\begin{aligned} R_n^\star(\varepsilon) = \inf_{q(z|X_{1:n},C)} \Big\{ & I(X_{1:n}; Z \mid C) : \\ & \inf_{p_\psi} \mathbb{E}[-\log p_\psi(X_{1:n} \mid Z, C)] \le \varepsilon \Big\}. \end{aligned} \tag{40}$$

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

$$\Delta_n = C_n - R_n^\star(\varepsilon). \tag{41}$$

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 $p_i(z) = p_\psi(x_i \mid x_{<i}, z, c)$, and let $\tau, \tau_i \in (0, 1]$ denote probability thresholds. Define

$$\mathcal{A}_{1:n}(\tau) = \left\{ z : \prod_{i=1}^{n} p_i(z) \ge \tau \right\}, \qquad \mathcal{A}_i(\tau_i) = \{ z : p_i(z) \ge \tau_i \}. \tag{42}$$

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

$$\mathcal{B}_n := \bigcap_{i=1}^{n} \mathcal{A}_i(\tau_i) \subseteq \mathcal{A}_{1:n}\left(\prod_{i=1}^{n} \tau_i \right). \tag{43}$$

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

$$\mathcal{B}_{n+1} = \mathcal{B}_n \cap \mathcal{A}_{n+1}(\tau_{n+1}) \subseteq \mathcal{B}_n. \tag{44}$$

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 $\ell_i(z) = -\log p_i(z)$,

$$\mathcal{L}_{\text{dec}}^{(n)}(z) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(z). \tag{45}$$

Its local Hessian is

$$H_n(z) = \nabla_z^2 \mathcal{L}_{\text{dec}}^{(n)}(z) = \frac{1}{n} \sum_{i=1}^{n} \nabla_z^2 \ell_i(z). \tag{46}$$

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,

$$\begin{aligned}\mathcal{L}_{\mathrm{dec}}^{(n)}(z+\delta) - \mathcal{L}_{\mathrm{dec}}^{(n)}(z) = \nabla \mathcal{L}_{\mathrm{dec}}^{(n)}(z)^\top \delta + \frac{1}{2}\delta^\top H_n(z)\delta \\ + o(\|\delta\|_2^2).\end{aligned} \tag{47}$$

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

$$r_n^{\mathrm{quad}}(\epsilon_d) \simeq \sqrt{\frac{2\epsilon_d}{\|H_n(z)\|_2}}. \tag{48}$$

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

$$\delta\phi_{\mathrm{FM}} = -\eta\lambda_{\mathrm{FM}} g_{\mathrm{FM}}. \tag{49}$$

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

$$\begin{aligned}\delta z_{\mathrm{FM}} &\approx J_{z,\phi}\delta\phi_{\mathrm{FM}} \\ &= -\eta\lambda_{\mathrm{FM}} J_{z,\phi} J_{z,\phi}^\top \nabla_{z_0}\ell_v.\end{aligned} \tag{50}$$

For batch updates, g_{FM} should be replaced by the corresponding aggregated parameter gradient. Fixing the current decoder and setting $z = z_0$, Eq. equation 47 gives

$$\begin{aligned}\Delta\mathcal{L}_{\mathrm{dec}}^{\mathrm{FM}} &:= \mathcal{L}_{\mathrm{dec}}^{(n)}(z+\delta z_{\mathrm{FM}}) - \mathcal{L}_{\mathrm{dec}}^{(n)}(z) \\ &\approx \nabla\mathcal{L}_{\mathrm{dec}}^{(n)}(z)^\top \delta z_{\mathrm{FM}} + \frac{1}{2}\delta z_{\mathrm{FM}}^\top H_n(z)\delta z_{\mathrm{FM}}.\end{aligned} \tag{51}$$

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:

$$\nabla_\phi\mathcal{L}_{\mathrm{joint}} \xrightarrow{\text{full detach}} \nabla_\phi\mathcal{L}_{\mathrm{rep}}, \qquad \nabla_\theta\mathcal{L}_{\mathrm{joint}} = \lambda_{\mathrm{FM}}\nabla_\theta\mathcal{L}_{\mathrm{FM}}. \tag{52}$$

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

Lemma D.1 (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.*

To prove Lemma D.1 and analyze the relation between the Stage 1 latent noise and the Stage 2 diffusion noise, we write the clean latent in the simplified form

$$Z = G + N_0, \tag{53}$$

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

$$Z_t = \alpha_t Z + \sigma_t \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I). \tag{54}$$

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

$$I(G; Z_t) = \frac{1}{2} \log \det \left[I + \alpha_t^2 \Sigma_G (\alpha_t^2 \Sigma_0 + \sigma_t^2 I)^{-1} \right]. \tag{55}$$

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

$$I(G; Z_t) = \frac{1}{2} \log \det \left[I + \Sigma_G (\Sigma_0 + e^{-\lambda_t} I)^{-1} \right]. \tag{56}$$

Eq. equation 56 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 4.3.

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 $A \in \{\mathrm{cond}, \mathrm{noise}, \mathrm{drop}\}$. For each condition, Eq. equation 2 in the main text holds:

$$H(Y \mid C) = I(Y; Z_A \mid C) + H(Y \mid C, Z_A). \tag{57}$$

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 $p_\psi(Y \mid C, Z_A)$, the cross-entropy decomposes exactly as

$$\begin{aligned} \mathcal{L}_A(\psi) = {} & H(Y \mid C, Z_A) \\ & + \mathbb{E}\, \mathrm{KL}(q(Y \mid C, Z_A) \| p_\psi(Y \mid C, Z_A)). \end{aligned} \tag{58}$$

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

$$Z_u = (1-u)Z + u\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I). \tag{59}$$

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

$$\mathbb{E}[\|\Delta Z_u\|_2^2 \mid Z, u] = u^2(d_z + \|Z\|_2^2). \tag{60}$$

A small u controls the mean-squared perturbation scale, but Gaussian noise does not have finite support. For each u, the decoder receives information $I(Y; Z_u \mid C)$; if a stronger perturbation constitutes a more degraded stochastic channel, this mutual information does not increase, and the corresponding $H(Y \mid C, Z_u)$ 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

$$f_\gamma = f_{\mathrm{u}} + \gamma\big(f_{\mathrm{c}} - f_{\mathrm{u}}\big), \tag{61}$$

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

$$\begin{aligned} s_{\mathrm{c}}(x,u) &= \nabla_x \log p_u(x \mid \mathcal{K}, A), \\ s_{\mathrm{u}}(x,u) &= \nabla_x \log p_u(x \mid \mathcal{K}). \end{aligned} \tag{62}$$

By Bayes' rule,

$$p_u(x \mid \mathcal{K}, A) = \frac{p_u(A \mid x, \mathcal{K}) p_u(x \mid \mathcal{K})}{p(A \mid \mathcal{K})}, \tag{63}$$

and taking the gradient with respect to x yields

$$s_{\mathrm{c}}(x,u) - s_{\mathrm{u}}(x,u) = \nabla_x \log p_u(A \mid x, \mathcal{K}). \tag{64}$$

The standard CFG score therefore satisfies

$$\begin{aligned} s_\gamma &= s_{\mathrm{u}} + \gamma(s_{\mathrm{c}} - s_{\mathrm{u}}) \\ &= \nabla_x \log p_u(x \mid \mathcal{K}) + \gamma \nabla_x \log p_u(A \mid x, \mathcal{K}) \\ &= \nabla_x \log \widetilde{p}_{\gamma,u}(x \mid \mathcal{K}, A), \end{aligned} \tag{65}$$

where the unnormalized guided distribution is

$$\begin{aligned} \widetilde{p}_{\gamma,u}(x \mid \mathcal{K}, A) &\propto p_u(x \mid \mathcal{K})\, p_u(A \mid x, \mathcal{K})^\gamma \\ &\propto p_u(x \mid \mathcal{K}, A)^\gamma p_u(x \mid \mathcal{K})^{1-\gamma}. \end{aligned} \tag{66}$$

When the normalizing constant is finite, it is independent of x and does not affect the score. For a fixed u, in Eq. equation 66, $\gamma = 0$ 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. equation 9, is still valid. Consider a general Gaussian corruption path

$$X_u = \alpha_u X_0 + \sigma_u \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I), \tag{67}$$

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

$$s_Q(x, u) = \nabla_x \log p_u(x \mid Q). \tag{68}$$

The score identity for Gaussian convolution gives

$$\mathbb{E}[\varepsilon \mid X_u = x, Q] = -\sigma_u s_Q(x, u). \tag{69}$$

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

$$\mathbb{E}[X_0 \mid X_u = x, Q] = \frac{x + \sigma_u^2 s_Q(x, u)}{\alpha_u}. \tag{70}$$

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

$$\begin{aligned} v_Q(x, u) &= \mathbb{E}[\dot{\alpha}_u X_0 + \dot{\sigma}_u \varepsilon \mid X_u = x, Q] \\ &= \frac{\dot{\alpha}_u}{\alpha_u} x + \sigma_u \left(\frac{\dot{\alpha}_u \sigma_u}{\alpha_u} - \dot{\sigma}_u \right) s_Q(x, u). \end{aligned} \tag{71}$$

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

$$v_Q(x, u) = a_u x + b_u s_Q(x, u), \tag{72}$$

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

$$\begin{aligned} v_{\mathrm{u}} + \gamma_{\mathrm{gen}}(v_{\mathrm{c}} - v_{\mathrm{u}}) &= a_u x + b_u \left[s_{\mathrm{u}} + \gamma_{\mathrm{gen}}(s_{\mathrm{c}} - s_{\mathrm{u}}) \right] \\ &= a_u x + b_u s_\gamma. \end{aligned} \tag{73}$$

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

$$\alpha_u = 1 - u, \qquad \sigma_u = u, \tag{74}$$

Eq. equation 71 further reduces to

$$v_Q(x, u) = -\frac{x}{1 - u} - \frac{u}{1 - u} s_Q(x, u), \qquad 0 < u < 1, \tag{75}$$

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 $v_\gamma = v_{\mathrm{u}} + \gamma(v_{\mathrm{c}} - v_{\mathrm{u}})$. If the normalized density in Eq. equation 66 were also the actual marginal of this flow, it would have to satisfy the continuity equation

$$\partial_u \widetilde{p}_{\gamma,u} + \nabla_x \cdot \left(\widetilde{p}_{\gamma,u} v_\gamma \right) = 0. \tag{76}$$

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

$$\begin{aligned} \ell_{11}(y) &= \ell(y \mid \mathcal{U}, \mathcal{P}, H^Z), \quad \ell_{10}(y) = \ell(y \mid \mathcal{U}, \mathcal{P}, 0), \\ \ell_{01}(y) &= \ell(y \mid \mathcal{U}, \varnothing, H^Z), \quad \ell_{00}(y) = \ell(y \mid \mathcal{U}, \varnothing, 0). \end{aligned} \tag{77}$$

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

$$\begin{aligned}\ell_{\gamma_{\mathrm{p}},\gamma_{\mathrm{z}}} = \gamma_{\mathrm{p}}\gamma_{\mathrm{z}}\ell_{11} + \gamma_{\mathrm{p}}(1-\gamma_{\mathrm{z}})\ell_{10} \\ + (1-\gamma_{\mathrm{p}})\gamma_{\mathrm{z}}\ell_{01} + (1-\gamma_{\mathrm{p}})(1-\gamma_{\mathrm{z}})\ell_{00}.\end{aligned} \tag{78}$$

This bilinear combination is independent of the nesting order of the two axes. Setting $\gamma_{\mathrm{p}} = 1$ modulates only the latent, $\gamma_{\mathrm{z}} = 1$ modulates only the prefix, and $\gamma_{\mathrm{z}} = 0$ 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. equation 78, and $p_{ab} = \mathrm{softmax}(\ell_{ab})$ the corresponding token probability. For any candidate token y,

$$\ell_{ab}(y) = \log p_{ab}(y) + \kappa_{ab}, \tag{79}$$

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

$$\sum_{a,b\in\{0,1\}} c_{ab} = 1, \qquad p_{\gamma_{\mathrm{p}},\gamma_{\mathrm{z}}}(y) \propto \prod_{a,b\in\{0,1\}} p_{ab}(y)^{c_{ab}}. \tag{80}$$

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

$$\delta_t = 1 - \cos(\mathrm{LN}(h_t), \mathrm{LN}(h_{t+1})), \qquad t = 1, \ldots, T-1, \tag{81}$$

and is mapped to a boundary probability by

$$p_t = \sigma(\alpha(\delta_t - \tau_0) + b), \qquad \mathrm{boundary}_t = \mathbb{1}[p_t > 0.5], \tag{82}$$

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 $\rho^\star$ and the measured rate on the current global batch is $\hat{\rho}$, then b does not participate in backpropagation and is updated as

$$b \leftarrow \mathrm{clip}(b - \eta_b(\hat{\rho} - \rho^\star), -c_b, c_b), \tag{83}$$

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. equation 82, a hard boundary satisfies

$$p_t > \frac{1}{2} \quad \Longleftrightarrow \quad \delta_t > \tau_{\text{eff}}(b) := \tau_0 - \frac{b}{\alpha}. \tag{84}$$

Thus, increasing b lowers the effective threshold and increases the boundary density; together with Eq. equation 83, the sign of b forms a negative-feedback loop. With the encoder and data distribution fixed, define

$$r(b) := \mathbb{E}[\hat{\rho} \mid b] \tag{85}$$

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 $\rho^\star$ 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 $b^\star \in (-c_b, c_b)$ such that $r(b^\star) = \rho^\star$. If r is differentiable in a neighborhood $\mathcal{I}$ of $b^\star$ and satisfies*

$$0 < m \le r'(b) \le L < \infty, \qquad b \in \mathcal{I}, \tag{86}$$

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

$$F(b) = \Pi_{[-c_b, c_b]}\big[b - \eta_b\big(r(b) - \rho^\star\big)\big] \tag{87}$$

is a contraction in a sufficiently small neighborhood of $b^\star$, with

$$|b_{n+1} - b^\star| \le q|b_n - b^\star|, \qquad q = \max\{|1 - \eta_b m|, |1 - \eta_b L|\} < 1. \tag{88}$$

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

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

$$r(b_n) - r(b^\star) = r'(\xi_n) e_n. \tag{89}$$

Ignoring the projection, the mean-response update in Eq. equation 87 gives

$$e_{n+1} = \big(1 - \eta_b r'(\xi_n)\big) e_n. \tag{90}$$

From Eq. equation 86 and $0 < \eta_b < 2/L$, we have $|1 - \eta_b r'(\xi_n)| \le q < 1$. The interval projection $\Pi_{[-c_b, c_b]}$ is non-expansive, and since $b^\star$ lies in the interior of the interval, the projection does not enlarge the error. This yields Eq. equation 88 and completes the proof. □

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, $r_n(b) = \mathbb{E}[\hat{\rho}_n \mid b, \phi_n]$ the actual response, and $b_n^\star$ its equilibrium satisfying $r_n(b_n^\star) = \rho^\star$. If the response at every step has a uniform contraction factor $q < 1$ over the visited region, then

$$|b_{n+1} - b_{n+1}^\star| \le q|b_n - b_n^\star| + \eta_b|\varepsilon_n| + |b_{n+1}^\star - b_n^\star|, \tag{91}$$

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:

$$M_{ij}^{\text{pool}} = \mathbb{1}[g(i) = g(j)], \qquad r_k = \frac{1}{L_k} \sum_{t \in \mathcal{S}_k} \widetilde{h}_t, \tag{92}$$

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

$$[\mu_k, \log \sigma_k^2] = W_{\text{post}} \bar{r}_k. \tag{93}$$

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,

$$p(L_k \mid z_k) = \text{softmax}(\text{MLP}_{\text{dur}}(g_k)). \tag{94}$$

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

$$h_t^Z = W_c \left[g_{g(t)} \,\middle\|\, \text{MLP}_{\text{loc}}\left(\frac{j}{L}, \frac{L-j}{L}, \log L \right) \right]. \tag{95}$$

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

$$\text{Allow}(i, j) = \begin{cases} \text{cl}(j) \wedge [\beta(j) \le \beta(i)], & \text{cl}(i) = 1, \\ \big(\text{cl}(j) \wedge [\beta(j) < \beta(i)]\big) \vee \big(\neg\, \text{cl}(j) \wedge [\beta(j) = \beta(i)]\big), & \text{cl}(i) = 0. \end{cases} \tag{96}$$

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,

$$\mathcal{L}_{\text{MDM}}^A = \frac{1}{|\mathcal{M}_A|} \sum_{t \in \mathcal{M}_A} \frac{-\log p_\psi(y_t \mid \widetilde{Y}, H_A^Z)}{\pi_{\beta(t)}}, \qquad A \in \{\textsf{cond}, \textsf{noise}, \textsf{drop}\}, \tag{97}$$

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:

$$\mathcal{L}_{\text{encmask}} = -\frac{1}{|\mathcal{M}_{\text{e}}|} \sum_{t \in \mathcal{M}_{\text{e}}} \log p_\psi(y_t \mid \widetilde{Y}, Z^{\text{encmask}}). \tag{98}$$

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

$$\widetilde{z}_k = \begin{cases} z_k, & A_k = \mathsf{cond}, \\ (1-u_k^{\mathrm{c}})z_k + u_k^{\mathrm{c}}\epsilon_k, & A_k = \mathsf{noise}, \\ \varnothing, & A_k = \mathsf{drop}, \end{cases} \qquad u_k^{\mathrm{c}} \in [0, u_{\max}], \quad \epsilon_k \sim \mathcal{N}(0, I). \tag{99}$$

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

$$\mathcal{L}_{\mathrm{bal}} = \lambda_{\mathrm{c}}\mathcal{L}_{\mathrm{tok}}^{\mathsf{cond}} + \lambda_{\mathrm{n}}\mathcal{L}_{\mathrm{tok}}^{\mathsf{noise}} + \lambda_{\mathrm{d}}\mathcal{L}_{\mathrm{tok}}^{\mathsf{drop}}, \tag{100}$$

where $\mathcal{L}_{\mathrm{tok}}^{A} = \mathcal{L}_{\mathrm{MDM}}^{A}$ as defined in Eq. equation 97.

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

$$\mathcal{L}_{\mathrm{gen}} = \mathcal{L}_{\mathrm{encmask}}. \tag{101}$$

The unified objective of the discrete generator decoder is therefore

$$\mathcal{L}_{\mathrm{DGD}} = \mathcal{L}_{\mathrm{bal}} + \lambda_{\mathrm{gen}}\mathcal{L}_{\mathrm{gen}}. \tag{102}$$

The key target of $\mathcal{L}_{\mathrm{gen}}$ 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

$$\mathrm{KL}_k = \frac{1}{2}\sum_{d=1}^{d_z}\left(\mu_{k,d}^2 + \sigma_{k,d}^2 - 1 - \log\sigma_{k,d}^2\right), \tag{103}$$

and the duration loss is

$$\mathcal{L}_{\mathrm{dur}} = -\frac{1}{K}\sum_{k=1}^{K}\log p_\psi(L_k \mid z_k). \tag{104}$$

Combined with Eq. equation 6, the total Stage 1 objective is

$$\mathcal{L}^{(1)} = \mathcal{L}_{\mathrm{DGD}} + \lambda_{\mathrm{dur}}\mathcal{L}_{\mathrm{dur}} + \beta_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}. \tag{105}$$

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. equation 105. Let Z be the sampled posterior latent of the current online encoder. The DiT reads only

$$\bar{Z} = \mathrm{sg}(Z), \tag{106}$$

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. equation 7 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

$$\mathrm{Allow}_{\mathrm{DiT}}(i,j) = \mathbb{1}[\beta(j) < \beta(i)]\,\mathrm{cl}(j) \;\vee\; \mathbb{1}[\beta(j) = \beta(i)]\neg\,\mathrm{cl}(j), \tag{107}$$

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 $c \in \{1, \ldots, B-1\}$ 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,

$$\nabla_\phi \mathcal{L}_{\mathrm{FM}} = 0, \qquad \nabla_\theta \mathcal{L}_{\mathrm{FM}} \neq 0, \tag{108}$$

and

$$\mathcal{L}^{(2)} = \mathcal{L}^{(1)} + \lambda_{\mathrm{FM}} \mathcal{L}_{\mathrm{FM}}. \tag{109}$$

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

$$K_{\mathrm{p}} = Bq + r, \qquad 0 \le r < B. \tag{110}$$

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:

$$Z_j^{(0)} = \left[z^{\mathrm{p}}_{K_{\mathrm{p}}-r+1:K_{\mathrm{p}}} \,\middle\|\, \epsilon_1, \ldots, \epsilon_{B-r} \right], \qquad \epsilon_i \sim \mathcal{N}(0, I). \tag{111}$$

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,

$$\hat{v} = v_{\mathrm{u}} + \gamma_{\mathrm{gen}}(v_{\mathrm{c}} - v_{\mathrm{u}}), \tag{112}$$

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,

$$\widehat{L}_k = \arg\max_L p_\psi(L \mid z_k), \tag{113}$$

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. equation 10. 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:

$$Z_j^{\mathrm{corr}} = \mu_\phi\Big(Y_j; \widehat{L}_{\mathcal{B}_j}, \mathcal{K}^E_{<j}\Big). \tag{114}$$

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

$$\begin{aligned} \mathcal{K}^{\mathrm{DiT}}_j &= Z^{\mathrm{corr}}_{\le j}, \\ \mathcal{K}^{\mathrm{cond}}_j &= \mathrm{Cond}(Z^{\mathrm{corr}}_{\le j}), \\ \mathcal{K}^{E}_j &= \mathrm{EncState}(Y_{\le j}), \\ \mathcal{K}^{\mathrm{tok}}_j &= Y^{\mathrm{clean}}_{\le j}. \end{aligned} \tag{115}$$

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,

$$\mathcal{K}^Z_{\le j} = E_\phi\Big(\mathcal{K}^Y_{\le j}; \widehat{\mathcal{S}}_{\le j}\Big), \tag{116}$$

rather than $\mathcal{K}^Z_{\le j} = Z^{\mathrm{prop}}_{\le j}$. 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 Appendix F.3. The refined condition is prepared only once per block:

$$H^Z_j = \mathrm{Cond}\Big(\mathrm{Expand}(Z^{\mathrm{prop}}_j, \widehat{L}_{\mathcal{B}_j}); \mathcal{K}^{\mathrm{cond}}_{<j}\Big), \qquad H^{Z,(b)}_j = bH^Z_j, \tag{117}$$

Algorithm 1 Unified block-wise inference of Cola2

Require: Prompt C, latent block size B, DiT steps S_{gen}
1: $z^{\text{p}}_{1:K_{\text{p}}} \leftarrow \mu(E_\phi(C))$
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** $i = 0, \ldots, S_{\text{gen}} - 1$ **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. equation 10
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

where $\mathcal{K}^{\text{cond}}_{<j}$ is the committed condition-stack history and $\widehat{L}_{\mathcal{B}_j}$ 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 $\mathcal{Q}_j = \{N_{\text{k}} - N_{\text{q}} + 1, \ldots, N_{\text{k}}\}$. The two prefix visibilities correspond to

$$m_r^{(a)} = m_r\big(a + (1-a)\mathbb{1}[r \in \mathcal{Q}_j]\big), \qquad \text{pos}_r^{(a)} = \max\left(\sum_{s=1}^{r} m_s^{(a)} - 1, 0\right). \tag{118}$$

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, $s \in \{0, \ldots, M_j - 1\}$ 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. equation 78. Denoting the combined logits at position t by $\ell_t^{(s)}$, the candidate token and confidence under greedy decoding are

$$\widehat{y}_t^{(s)} = \arg\max_y \ell_t^{(s)}(y), \qquad c_t^{(s)} = \log \text{softmax}(\ell_t^{(s)})\big(\widehat{y}_t^{(s)}\big), \quad t \in \mathcal{M}_s. \tag{119}$$

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` ($\gamma_{\mathrm{z}} - 1$) and `mdm_decoder_prefix_cfg_weight` ($\gamma_{\mathrm{p}} - 1$). The prefix parameter defaults to 0, corresponding to $\gamma_{\mathrm{p}} = 1$. 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 derives the computational cost according to the training and inference execution graphs of Cola2. There are two key differences from Cola DLM (Guo et al., 2026): 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 accounting of Cola DLM as "Stage 1 VAE + Stage 2 DiT/decoder" or a one-way "DiT $\rightarrow$ decoder" does not apply to Cola2.

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 $\mathcal{A}_R^{\mathrm{exec}}$ the cumulative number of executed query–key pairs per layer. When all layers share the same shape,

$$\begin{aligned} \mathcal{C}_{R,\mathrm{backbone}}^{\mathrm{fwd}} &\approx 2N_R Q_R + 4n_R d_R^{\mathrm{q}} \mathcal{A}_R^{\mathrm{exec}}, \\ \mathcal{C}_{R,\mathrm{backbone}}^{\mathrm{train}} &\approx 6N_R Q_R + 12n_R d_R^{\mathrm{q}} \mathcal{A}_R^{\mathrm{exec}}, \end{aligned} \tag{120}$$

where the attention term includes both $QK^\top$ and the attention–value product; heterogeneous layers are summed layer by layer. Let $\mathcal{A}_R^{\mathrm{valid}}$ 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

$$\mathcal{C}_{\mathrm{head}}^{\mathrm{fwd}} \approx 2V d_{\mathrm{out}} Q_{\mathrm{logit}}, \qquad \mathcal{C}_{\mathrm{head}}^{\mathrm{train}} \approx 6V d_{\mathrm{out}} Q_{\mathrm{logit}}. \tag{121}$$

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,

$$K = \rho T. \tag{122}$$

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

$$P \approx \frac{B}{\rho_{\text{gen}}}, \qquad J \approx \frac{\rho_{\text{gen}} T_{\text{new}}}{B}. \tag{123}$$

Duration prediction does not guarantee $\rho_{\text{gen}} = \rho^\star$. 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,

$$\begin{aligned}\mathcal{C}_{\text{rep}}^{\text{MDM}}(T, K) \approx{}& 2\,\mathcal{C}_E(T, K) + 2\,\mathcal{C}_D^{\text{MDM}}(2T) \\ &+ \mathcal{C}_{\text{cond/dur/KL}}^{\text{MDM}},\end{aligned} \tag{124}$$

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 $24N_DT$, 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

$$c_{\text{rep}}^{(s)} := \frac{\mathbb{E}_s[\mathcal{C}_{\text{rep}}(T, K)]}{\mathbb{E}_s[T]}, \qquad s \in \{1, 2\}. \tag{125}$$

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, $K_{\text{c}} = K - r_{\text{last}}$, and we define

$$\chi_G = \frac{K + K_{\text{c}}}{K}, \qquad 1 \le \chi_G < 2. \tag{126}$$

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

$$\mathcal{C}_{G,\text{linear}}^{\text{train}}(T, K) \approx 6N_G(K + K_{\text{c}}) = 6\chi_G \rho N_G T. \tag{127}$$

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:

$$\mathcal{A}_G^{\text{valid}} = B^2 \left(\sum_{j=1}^{J_{\text{tr}}-1} j + \sum_{j=1}^{J_{\text{tr}}} j \right) = K^2. \tag{128}$$

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

$$\mathcal{C}^{\text{train}}_{G,\text{attn}} \approx 12 n_G d_G^{\text{q}} \mathcal{A}_G^{\text{exec}}, \qquad \mathcal{A}_G^{\text{valid}} = \mathcal{O}(\rho^2 T^2), \tag{129}$$

where n_G and d_G^{q} follow the module-level notation. Let $\mathcal{C}^{\text{train}}_{G,\text{aux}}$ 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

$$c_G^{(2)} = \frac{\mathbb{E}_2[\mathcal{C}^{\text{train}}_{G,\text{linear}} + \mathcal{C}^{\text{train}}_{G,\text{attn}} + \mathcal{C}^{\text{train}}_{G,\text{aux}}]}{\mathbb{E}_2[T]}. \tag{130}$$

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

$$\mathcal{C}^{\text{train}}_{\text{Cola2}} \approx D_1 c_{\text{rep}}^{(1)} + D_2 \big(c_{\text{rep}}^{(2)} + c_G^{(2)}\big). \tag{131}$$

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

$$r_{\text{rep}}^{(s)} = \frac{c_{\text{rep}}^{(s)}}{c_{\text{AR}}^{\text{train}}}, \qquad r_G^{\text{train}} = \frac{c_G^{(2)}}{c_{\text{AR}}^{\text{train}}}. \tag{132}$$

Then

$$\begin{aligned} R_{\text{train}} &:= \frac{\mathcal{C}^{\text{train}}_{\text{Cola2}}}{D_{\text{AR}} c_{\text{AR}}^{\text{train}}} \\ &\approx \frac{D_1}{D_{\text{AR}}} r_{\text{rep}}^{(1)} + \frac{D_2}{D_{\text{AR}}} \big(r_{\text{rep}}^{(2)} + r_G^{\text{train}}\big). \end{aligned} \tag{133}$$

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

$$\text{DiT proposal} \to \text{DGD token decoding} \to \text{fixed-span re-encoding} \to \text{corrected-cache commit}. \tag{134}$$

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

$$\begin{aligned} Q_G^{\text{prop}} &= \sum_{j=1}^{J} B_j^{\text{exec}} \sum_{i=1}^{S_{\text{gen}}} \nu_{G,j,i} \\ &\approx \rho_{\text{gen}} \nu_{\text{gen}} S_{\text{gen}} T_{\text{new}}. \end{aligned} \tag{135}$$

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

$$\kappa_G^{\text{commit}} := \frac{\mathcal{C}^{\text{commit}}_{G,\text{linear}}}{2 N_G K_{\text{new}}}, \tag{136}$$

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

$$\begin{aligned}\mathcal{C}_G^{\text{infer}} \approx 2N_G Q_G^{\text{prop}} + \mathcal{C}_{G,\text{linear}}^{\text{commit}} \\ + \mathcal{C}_{G,\text{attn}}^{\text{infer}} + \mathcal{C}_{G,\text{aux}}^{\text{infer}},\end{aligned} \tag{137}$$

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. equation 120.

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

$$\begin{aligned}\mathcal{C}_D^{\text{MDM}} \approx 2N_D \sum_{j=1}^{J} \left(W_j^{\text{exec}} \sum_{i=1}^{M_j} \nu_{D,j,i} + W_j^{\text{com}} \right) \\ + \mathcal{C}_{D,\text{attn}}^{\text{MDM}} + \mathcal{C}_{\text{head/interface}}^{\text{MDM}}.\end{aligned} \tag{138}$$

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

$$\begin{aligned}\mathcal{A}_D^{\text{valid}} = \sum_{j=1}^{J} \Bigg[\sum_{i=1}^{M_j} \left(\nu_{j,i}^{\text{on}} W_j P_j^Y + \nu_{D,j,i} W_j^2 \right) \\ + W_j (P_j^Y + W_j) \Bigg].\end{aligned} \tag{139}$$

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 $\mathcal{A}_D^{\text{exec}}$:

$$\mathcal{C}_{D,\text{attn}}^{\text{MDM}} \approx 4 n_D d_D^{\text{q}} \mathcal{A}_D^{\text{exec}}. \tag{140}$$

$\mathcal{C}_{\text{head/interface}}^{\text{MDM}}$ 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

$$\begin{aligned}\mathcal{C}_{\text{fb,enc}} \approx 2N_{F,t} Q_{F,t} + 2N_{F,z} Q_{F,z} \\ + \mathcal{C}_{\text{fb,attn}} + \mathcal{C}_{\text{fb,proj}},\end{aligned} \tag{141}$$

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 $\mathcal{C}_{\mathrm{cond,commit}}$, we have $\mathcal{C}_{\mathrm{fb}} = \mathcal{C}_{\mathrm{fb,enc}} + \mathcal{C}_{\mathrm{cond,commit}}$. 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 $\mathcal{C}_G^{\mathrm{infer}}$.

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, $\mathcal{A}_{\mathrm{AR}}^{\mathrm{exec}}$ the executed attention pairs, and $\mathcal{C}_{\mathrm{head,AR}}^{\mathrm{infer}}$ the output-projection cost. Then

$$\begin{aligned}\mathcal{C}_{\mathrm{AR}}^{\mathrm{infer}} \approx\ & 2N_{\mathrm{AR}}Q_{\mathrm{AR}} + 4n_{\mathrm{AR}}d_{\mathrm{AR}}^{\mathrm{q}}\mathcal{A}_{\mathrm{AR}}^{\mathrm{exec}} \\ & + \mathcal{C}_{\mathrm{head,AR}}^{\mathrm{infer}}.\end{aligned} \tag{142}$$

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

$$r_G^{\mathrm{infer}} = \frac{\mathcal{C}_G^{\mathrm{infer}}}{\mathcal{C}_{\mathrm{AR}}^{\mathrm{infer}}}, \quad r_{\mathrm{DGD}} = \frac{\mathcal{C}_D^{\mathrm{infer}}}{\mathcal{C}_{\mathrm{AR}}^{\mathrm{infer}}}, \quad r_{\mathrm{fb}} = \frac{\mathcal{C}_{\mathrm{fb}}}{\mathcal{C}_{\mathrm{AR}}^{\mathrm{infer}}}, \tag{143}$$

so that

$$R_{\mathrm{infer}} = \frac{\mathcal{C}_G^{\mathrm{infer}} + \mathcal{C}_D^{\mathrm{infer}} + \mathcal{C}_{\mathrm{fb}}}{\mathcal{C}_{\mathrm{AR}}^{\mathrm{infer}}} = r_G^{\mathrm{infer}} + r_{\mathrm{DGD}} + r_{\mathrm{fb}}. \tag{144}$$

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

$$\frac{2N_G Q_G^{\mathrm{prop}} + \mathcal{C}_{G,\mathrm{linear}}^{\mathrm{commit}}}{2N_{\mathrm{AR}}T_{\mathrm{new}}} \approx \rho_{\mathrm{gen}}\big(\nu_{\mathrm{gen}}S_{\mathrm{gen}} + \kappa_G^{\mathrm{commit}}\big)\frac{N_G}{N_{\mathrm{AR}}}. \tag{145}$$

This is not the total inference ratio. When end-to-end initialization is included, the numerator and denominator of Eq. equation 144 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

$$\mathcal{H}_{\mathrm{Cola2}} \approx \sum_{j=1}^{J}\big(S_{\mathrm{gen}} + H_{D,j} + H_{F,j}\big). \tag{146}$$

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

$$\frac{\mathcal{H}_{\mathrm{Cola2}}}{\mathcal{H}_{\mathrm{AR}}} \approx \frac{\rho_{\mathrm{gen}}}{B}\big(S_{\mathrm{gen}} + H_D + H_F\big). \tag{147}$$

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

$$B > \rho_{\mathrm{gen}}\big(S_{\mathrm{gen}} + H_D + H_F\big), \tag{148}$$

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,

$$\tau_{\text{Cola2}} \approx \tau_{\text{init}} + \sum_{j=1}^{J} \left[\sum_{i=1}^{S_{\text{gen}}} \tau_{G,j,i} + \tau_{\text{prep},j} + \tau_{D,j} + \tau_{F,j} \right]. \tag{149}$$

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,

$$\begin{aligned} \text{Mem}_{\text{KV}} \approx 2 s_{\text{KV}} N_{\text{seq}} \Big[& (n_G d_G^{\text{KV}} + n_E d_E^{\text{KV}}) K_{\text{hist}} \\ & + (n_C d_C^{\text{KV}} + n_D d_D^{\text{KV}}) T_{\text{hist}} \Big]. \end{aligned} \tag{150}$$

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. equation 150 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 Additional Experimental Details and Results

I.1 Training and Evaluation Settings

Data and evaluation. All models are trained on the open-source OLMo 2 pretraining corpus (OLMo et al., 2024). 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. 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 scaling experiments report the accuracy on all five tasks and their average.

Baselines. The internal ablations compare different configurations of Cola2; in the experiments on the semantic latent space and the latent generator, CFG is used to fix the decoder in the latent-conditioned reconstruction mode. Cola-MDM has no sequence compression (latent-to-token ratio

1) and shares the selected MDM decoder design with Cola2-MDM, which separates the effect of decoder improvements from that of dynamic compression. All baselines and Cola2-MDM are strictly matched in model size (about 2.4B parameters) and use identical training data and evaluation protocols.

Optimization. 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.

Default configuration. 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 ($\mathrm{loc} = 2.0$, $\mathrm{scale} = 1.0$); 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 scaling experiments train on about 210B tokens.

I.2 Additional Results

VAE architecture. 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 11, left). An oversized encoder makes the latent distribution more complex and harder for the DiT to model. Intra-span bidirectional attention is the best pooling operator (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 11, right).

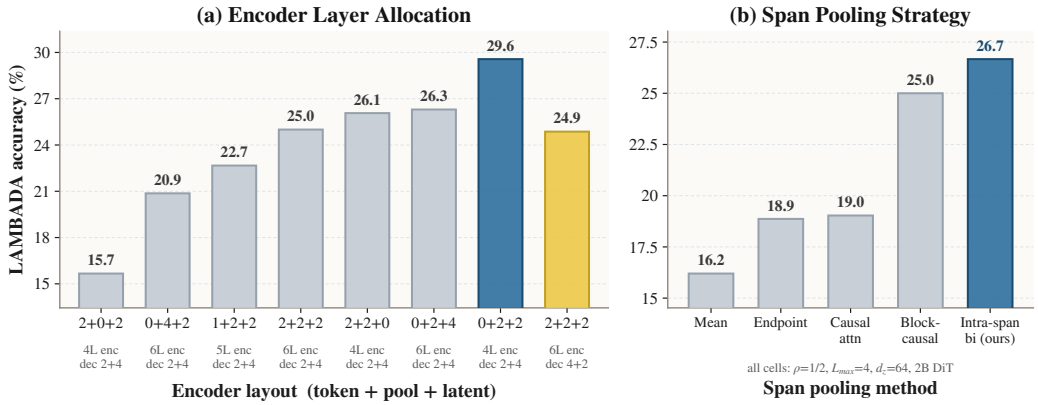


Figure 11: **VAE architecture.** Left: the allocation of encoder layers matters more than depth, with $0{+}2{+}2$ performing best. Right: intra-span bidirectional attention is the best pooling operator.

Generator guidance and inference steps. At 8 steps, accuracy rises from 12.2% at $\gamma_{\mathrm{gen}} = 1$ to 37.1% at $\gamma_{\mathrm{gen}} = 14$; at 16 steps, the peak moves to $\gamma_{\mathrm{gen}} = 20$ (38.5%) (Figure 12, left). Using 1 and 4 steps loses about 14 and 5 points, whereas results between 8 and 128 steps differ by at most 0.3 points (Figure 12, right); the benefit of more than 8 steps comes mainly from tolerating stronger guidance. Consistently, in the joint DiT/demasking sweep (Figure 9, left), increasing demasking from 1 to 8 steps yields 6.9 points with 8 DiT steps and 12.5 points with only 2 DiT steps, but still falls short of 8 DiT steps.

Maximum span length and truncation. Figure 13 explains the non-monotonic effect of $L_{\max}$ in Figure 3. When truncation is frequent, a smaller $L_{\max}$ keeps each latent simpler; once the cap mass

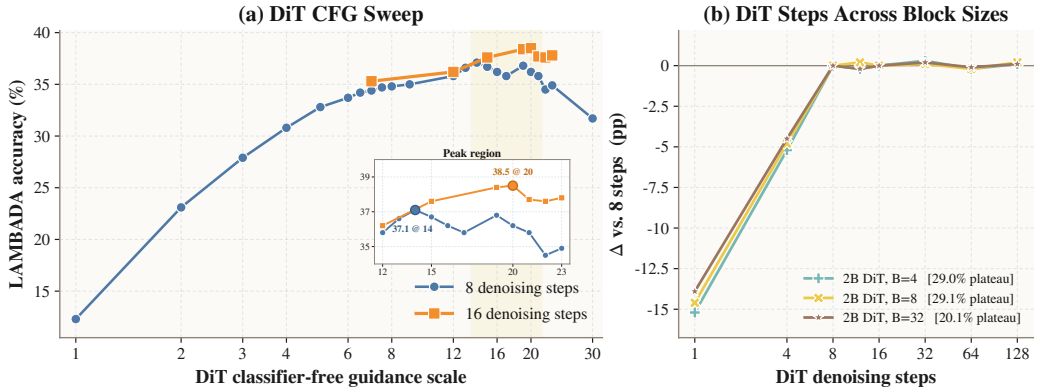


Figure 12: **DiT inference.** Left: generator CFG sweep. Right: the DiT saturates at 8 steps.

drops to about 1% (span length 4, $L_{\max} = 18$), preserving complete semantic spans dominates and accuracy rises to 21.7%.

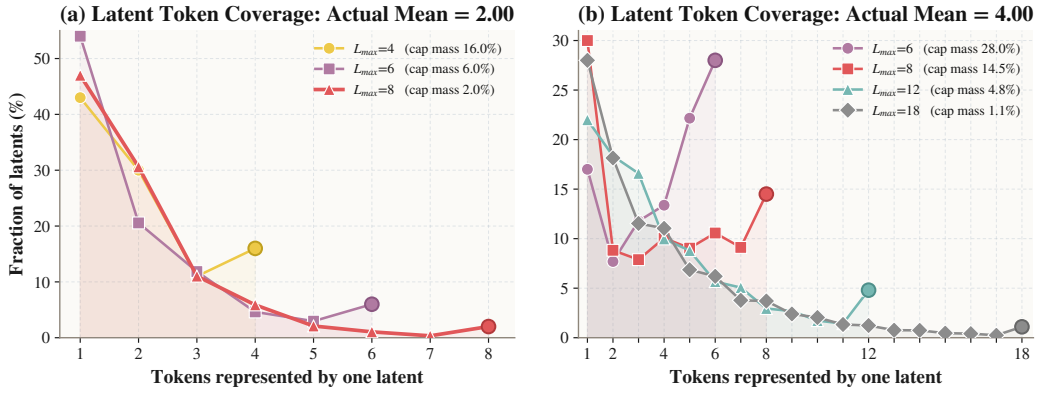


Figure 13: **Distribution of the number of tokens covered by each latent.** Increasing $L_{\max}$ reduces the number of forcibly truncated spans.

Decoder factorization. As shown in 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. Given a shared latent–token interface, the specific decoder factorization is therefore not the decisive factor, and the hierarchical framework applies to both.

Table 3: **Decoder factorization.** The two decoders perform similarly (LAMBADA, %).

Decoder	50k	100k	150k
AR-MTP decoder	**39.7**	43.7	44.1
MDM decoder	38.8	**44.4**	**45.5**
Diff. (MDM − AR-MTP)	−0.9	+0.7	+1.4

Decoder capacity. With the DiT fixed, increasing the decoder from 500M to 2B yields monotonic gains of 2.1–2.4 points (Figure 14), showing that the decoder is capacity-limited; under a fixed total budget, however, allocating more parameters to the DiT is preferable (Figure 8).

Decoder guidance in scaled training. The optimal w_{dec} lies mostly within $[-1.0, -0.8]$, consistent with the -0.9 selected in Section 4.4, and drifts slowly toward 0 during training (Figure 15). The

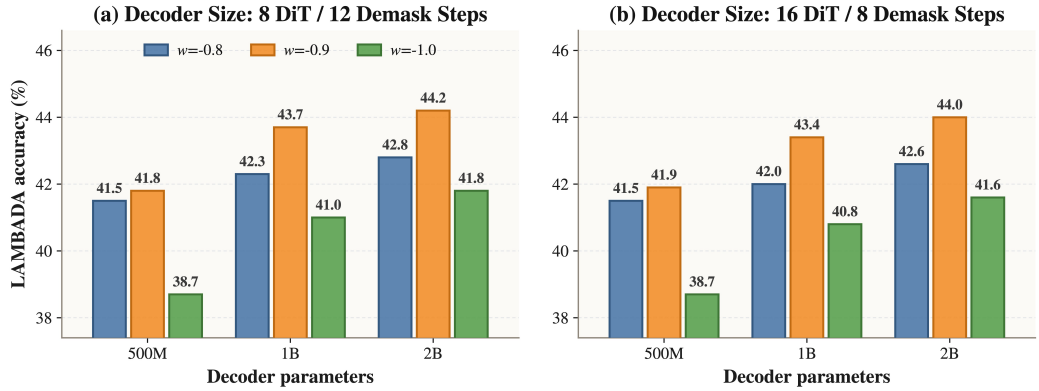


Figure 14: **Scaling the decoder with a fixed DiT.** Accuracy improves monotonically with decoder size.

drift of Cola2-MDM is more gradual (about $-0.9 \to -0.8$, versus $-0.95 \to -0.8$ for Cola-MDM), consistent with its latent distribution being harder to learn and converging more slowly.

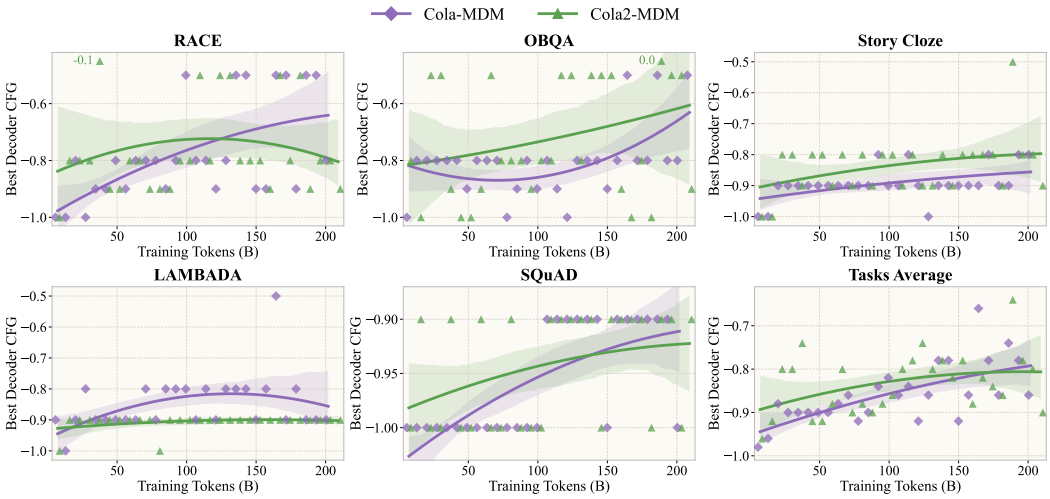


Figure 15: **Optimal decoder CFG weight during training.** For both models, the optimal weight remains within $[-1.0, -0.8]$ and drifts slowly toward 0.