# STRUCTURED PARALLELISM FOR DIFFUSION LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Discrete diffusion language models generate text by repeatedly updating noisy or masked tokens. At each step, a bidirectional Transformer backbone uses context from both directions to compute token representations, which an output layer converts into probabilities. Most models use a product distribution: they learn individual token predictions and sample them independently given the noisy sequence. This simplifies training and sampling, but even accurate individual predictions can yield inconsistent combinations. Later steps can improve consistency, at the cost of further Transformer passes. We introduce *Structured-Parallel Diffusion Language Models* (SPDLMs), which model dependencies among tokens proposed in the same step. At each denoising step, a lightweight decoder reuses a single Transformer evaluation of the noisy sequence to generate dependent tokens in parallel groups. Sequential computation is confined to this decoder, without reevaluating the Transformer within the step. Experiments on controlled pretraining and adaptation of existing diffusion language models show that this structured joint distribution improves generation with few denoising steps, reducing the need for costly backbone evaluations.

## 1 INTRODUCTION

Autoregressive Transformers (Vaswani et al., 2017; Radford et al., 2018; Touvron et al., 2023) generate a token before conditioning the next prediction on it. Caching avoids recomputing the prefix, but generation still requires a sequence of backbone evaluations. Diffusion language models instead update many positions at each denoising step. Their denoising network consists of a bidirectional Transformer backbone, which computes contextual representations of the noisy sequence, and an output head, which converts these representations into token predictions (Austin et al., 2021; Sahoo et al., 2024; Shi et al., 2024). This parallelism can reduce the number of sequential backbone evaluations and thereby lower generation latency. Masked and uniform models such as LLaDA (Nie et al., 2025; Zhu et al., 2026; 2025), Sumi (Ye et al., 2026), and DiffusionGemma (DiffusionGemma Team et al., 2026) share this pattern, as do block and set-based variants that change which positions are active (Arriola et al., 2025; Deschenaux & Gulcehre, 2026; Arriola & Kuleshov, 2026).

Most diffusion language models assume that the denoiser's output distribution factorizes across positions. Then, the bidirectional Transformer uses the whole noisy sequence to compute token representations, but the output distribution is a product of per-position predictions conditioned on that sequence. This factorization makes likelihood evaluation and sampling tractable, while excluding dependencies among tokens proposed in the same step. Consequently, even accurate individual predictions can therefore produce inconsistent combinations. Later denoising steps can improve consistency by processing the updated sequence, but each requires another Transformer evaluation. The challenge is therefore to model dependencies within a step while retaining the computational advantage of parallel generation.

We introduce *Structured-Parallel Diffusion Language Models* (SPDLMs), in which each token's predictive distribution can be conditioned on selected tokens already sampled in the same denoising step. A fixed forest organizes these dependencies: independent groups run in parallel, and predictions within each group follow the chosen structure. By varying the dependency structure, our framework interpolates between fully independent token predictions and complete dependence within each group.



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.

We implement these distributions with a lightweight decoder added to the Transformer backbone. This decoder uses a recursive one-slot head, which stores the conditioning context in a single compact cache state. After one backbone evaluation, it samples tokens in the order specified by the dependency structure, using cached information about previously sampled tokens to condition subsequent predictions. With no conditioning tokens, the head reduces to the original vocabulary head. Longer conditioning contexts are handled by repeatedly applying the same read and write operations. The additional sequential computation is therefore confined to the output decoder, without reevaluating the backbone within the step.

We evaluate SPDLMs through controlled pretraining on TinyStories and adaptation of pretrained GIDD and Sumi models. The TinyStories experiments examine how dependencies within a denoising step affect the trade-off between generation quality and latency, with the largest gains when generation uses few steps. On GIDD, enabling the structured head produces samples whose distribution more closely matches held-out text, using the same adapted checkpoint. On Sumi, SPDLM improves GSM8K accuracy over both a model adapted with independent token predictions and an adaptation of the probabilistic recoupling method CoDD (Li et al., 2026). These gains are obtained by training directly on the target data, without distillation from a teacher model.

**Organization of the paper.** We begin by introducing the generation schedules and output factorizations used in our framework in Section 2, and present the forest-structured output distributions in Section 3. Then, Section 4 describes the implementation, training, and sampling, together with the computational cost. Finally, we report the numerical experiments in Section 5, and conclude and discuss limitations of the approach in Section 6.

## 2 Background and Preliminaries

Language models operate on a finite context window of length $L$: the portion of the sequence they condition on at a given generation step. Denote $[L] = \{1, \dots, L\}$ for the positions in a context window. In the standard autoregressive setting, this window consists of the prompt together with all previously committed tokens. In diffusion models (stage generation), the context window may also include a partially generated canvas, whose entries may be revised over multiple stages. The prompt is the fixed conditioning prefix; the canvas is the evolving sequence of tentative values; and the active-set schedule determines which positions are updated across stages. We denote by $C_n$ the prompt, current canvas, and stage information (such as the diffusion time or the blocks updated) before stage $n$.

**Generation schedules.** In autoregressive generation, the model predicts one token at stage $n$, conditioned on the prompt and all previously committed tokens. The active set therefore contains a single position, $A_n^{\mathrm{AR}} = v_n$ with $v_n \in [L]$. The sampled token is committed before generation proceeds to the next position. In full diffusion, all (non-prompt) positions can be revised at each stage. The context window contains the prompt and the current tentative token values, and the model proposes updates conditioned on this full sequence. In unconditional generation, $A_n^{\mathrm{full}} = [L]$. Otherwise, the prompt remains fixed while the remaining positions evolve until the final step. Block diffusion sits between these two models. The context window again contains the prompt and the current tentative token values, but its non-prompt positions are partitioned into ordered blocks $\mathcal{B}_1, \dots, \mathcal{B}_J$. At each stage, only one block is active: $A_n^{\mathrm{block}} = \mathcal{B}_j$ for some block $j$. The model refines this block over several local refinement steps before committing its tokens and moving to the next block. Previously completed blocks remain fixed.

These schedules can be described by a sequence $\mathcal{A} = (A_1, \dots, A_N)$, where $A_n \subseteq [L]$ contains the positions updated at stage $n$ (see Fig. 1). An adaptive procedure may choose each set from the current context; $\mathcal{A}$ then denotes the resulting sequence, not the selection rule. The update mechanism and, for diffusion, the noise times are specified separately. Writing $\mathbf{y}_{A_n} = (y_v)_{v \in A_n}$ for the values at these active positions, the model proposes them from the conditional joint law $\mathbf{y}_{A_n} \sim p_\theta(\cdot \mid C_n; \mathcal{F}(A_n))$. Here $\mathcal{F}$ is a fixed factorization rule applied to each selected set, not an observed random variable. The conventional choice is the order-1, or product, structure

$$\mathcal{F}_1(A) = \bigsqcup_{v \in A} \{v\}, \qquad p_\theta(\mathbf{y}_A \mid C; \mathcal{F}_1(A)) = \prod_{v \in A} p_{\theta,v}(y_v \mid C). \tag{1}$$

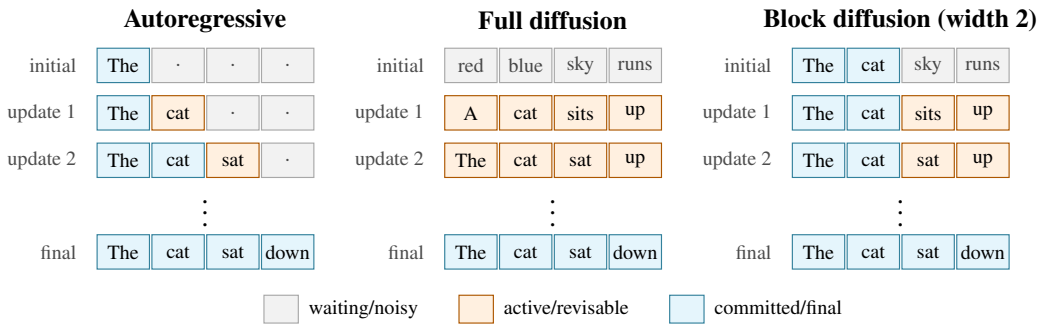



Figure 1: Canonical schedules with product within-stage structure. Autoregression commits one token per stage; full diffusion revises the full window for $N$ stages; block diffusion revises one block for $N_{\text{loc}}$ stages before committing it.

Each factor conditions on the same full context $C$, but the proposed values are conditionally independent given $C$. Thus $\mathcal{A}$ specifies which positions are updated together, whereas $\mathcal{F}$ controls their within-stage dependence. In autoregression, $|A_n| = 1$, so $\mathcal{F}_1$ is trivially exact within each stage and there is no nontrivial factorization to choose from. Full and block diffusion instead use $|A_n| > 1$, making $\mathcal{F}_1$ a genuine modeling restriction. Hence, diffusion schedules create nontrivial active sets by construction, motivating the study of conditional structures $\mathcal{F} \neq \mathcal{F}_1$ that exploit them without changing $\mathcal{A}$.

**The product restriction in parallel updates.** Consider a stage with active set $A$, context $C$, and target joint distribution $p^*(\mathbf{y}_A \mid C)$. The conventional update uses the product law in Eq. (1). Although the Transformer can coordinate its marginal logits through the shared context $C$, samples drawn from those logits remain conditionally independent. Consider the motivating example from CoDD (Li et al., 2026): a joint law with equal mass on "Los Angeles" and "Las Vegas." This yields the correct uniform marginals over $\{\text{Los}, \text{Las}\}$ and $\{\text{Angeles}, \text{Vegas}\}$, but the product law also assigns nonzero mass to invalid crossovers such as "Los Vegas" and "Las Angeles." Thus correct marginals do not determine the correct joint distribution. Later reverse stages can sometimes correct such errors by processing the updated canvas, but each of these additional stage requires another backbone evaluation. We instead keep $\mathcal{A}$ fixed and replace $\mathcal{F}_1$ with a normalized joint distribution that models a sparse set of dependencies. The approximation gap of the best product distribution for a single stage is derived in Section A.1.

### 2.1 Related Work

**Schedules and output laws.** Categorical and masked diffusion models make parallel token updates (Austin et al., 2021; Sahoo et al., 2024; Shi et al., 2024), including at scale (Nie et al., 2025; 2026). Block methods change the active positions and their order while retaining product predictions within an update (Arriola et al., 2025; Deschenaux & Gulcehre, 2026). CoDD couples these potentials through a tractable probabilistic circuit (Li et al., 2026); its joint law is therefore not a product. CoDiLA combines diffusion with local autoregression through a separate pretrained AR model, conditioned on embeddings averaged under the diffusion marginals (Hersche et al., 2026). Unlike CoDiLA, SPDLM uses a lightweight conditional head attached to the diffusion backbone, without a separate pretrained autoregressive model.

**Continuous and distilled few-step language models.** Continuous diffusion operates on embeddings rather than categorical transitions (Li et al., 2022; Chen et al., 2026; Hu et al., 2026). ELF generates without distillation at moderate budgets and uses progressive distillation for its extreme few-step results (Hu et al., 2026). D-MMD distills discrete transitions (Hoogeboom et al., 2026), while FMLM learns a two-time map from a pretrained continuous flow (Lee et al., 2026; Boffi et al., 2024).

**Graphical models.** Chow–Liu trees give KL-optimal tree approximations (Chow & Liu, 1968). The $t$-cherry construction extends them to $K$-variable cliques joined by $(K-1)$-variable separators (Kovács & Szántai, 2009). The decoder introduced in Section 4 implements this hierarchy conditional on the noisy backbone representation.

# 3 Structured Parallelism through Forest Factorizations

## 3.1 Conditioning on Clean Tokens

The simplest change to product prediction is to let a token depend on one clean token already sampled in the same update. Fix the noisy context $C$ and a component $T \subseteq A$ of $W$ positions. In Fig. 2, given four positions $v_1, \dots, v_4$, the token "York" is sampled first; then one can sample "New" and "City" in parallel conditioning on the first token; finally, "skyline" is sampled last conditioning on "City." The arrows specify conditional sampling order and not word order as the tokens are still associated with their position. In this example, the generated result reads as "New York City skyline." All factors see the same noisy context $C$ but differ in their additional clean information accessed.

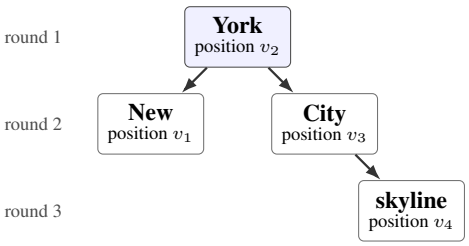



Figure 2: Order-2 tree: $W = 4, D = 3$.

A tree with root $v_\circ$ and parent map $\mathrm{pa}(v)$ defines the order-two law:

$$p_\theta(\mathbf{y}_T \mid C; \mathcal{F}_2(T)) = p_{\theta,v_\circ}(y_{v_\circ} \mid C) \prod_{v \in T \setminus \{v_\circ\}} p_{\theta,v}(y_v \mid y_{\mathrm{pa}(v)}, C). \tag{2}$$

A prediction may use several previously sampled clean tokens: write $\pi = (v_1, \dots, v_W)$ for the sampling order and call the clean conditioning set $S_{v_r} \subseteq \{v_1, \dots, v_{r-1}\}$ its separator. Order $K$ bounds this set to at most $K-1$ variables. The resulting law is

$$p_\theta(\mathbf{y}_T \mid C; \mathcal{F}_K(T)) = \prod_{r=1}^{W} p_{\theta,v_r}(y_{v_r} \mid \mathbf{y}_{S_{v_r}}, C), \quad \text{where } |S_{v_r}| \le K-1, \tag{3}$$

which is normalized because each factor conditions only on earlier variables. Empty separators recover $\mathcal{F}_1$. Our structured families follow the classical $t$-cherry construction (Kovács & Szántai, 2009); its additional clique constraints and approximation results are given in Section A.4 and Proposition A.2.

Our principal configurations use left-to-right paths (see e.g. Fig. 8). A strict $K = 2$ path sets $S_{v_r} = \{v_{r-1}\}$ for $r > 1$, whereas a full-prefix decoder uses $S_{v_r} = \{v_1, \dots, v_{r-1}\}$ and has $K = W$; the first separator is empty in both cases. The latter removes the graphical independence restriction within a component, although a compressed neural state can still limit its conditionals. The two paths have the same sampling order but retain different amounts of clean context.

## 3.2 Parallel Components Form the Forest

Partition the active set into disjoint components as $A_n = T_1 \sqcup \cdots \sqcup T_M$ such that for an equal width $W$ dividing $|A_n|$, we have $MW = |A_n|$. While each component has its own clean dependencies, they all condition on the global noisy context $C_n$, and the backbone context and active-set schedule $\mathcal{A}$ remain unchanged. The forest law is given by

$$p_\theta(\mathbf{y}_{A_n} \mid C_n; \mathcal{F}_K(A_n)) = \prod_{m=1}^{M} p_\theta(\mathbf{y}_{T_m} \mid C_n; \mathcal{F}_K(T_m)). \tag{4}$$

At round $d$, the positions $\mathcal{L}_d$ whose separators are available are sampled together (in parallel), across all components, where the number of rounds is the maximum component depth, $D = \max_m D_m$, rather than the number of active positions. Left-to-right paths have $D = W$ but branching trees can

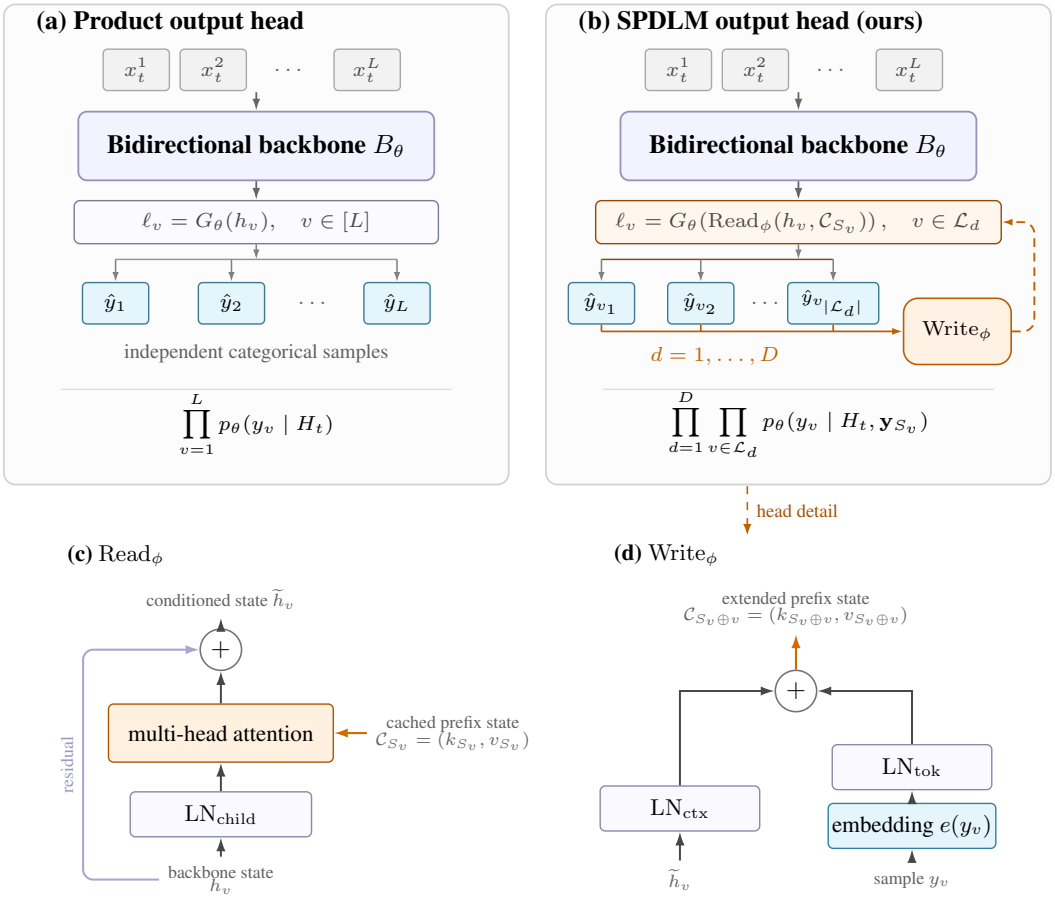



Figure 3: One backbone call, two output laws. Product decoding uses one independent round; SPDLM keeps $H_t$ fixed across $D$ lightweight structured rounds. Below, READ conditions a backbone state on its separator; WRITE is shown for full-prefix conditioning. For $K = 2$, its state input is the raw $h_v$ instead of $\widetilde{h}_v$.

be shallower: the example in Fig. 2 has $W = 4$, $K = 2$, and $D = 3$. Thus width, order, and depth describe different choices; Section A.5 gives the general construction and topology examples. The same rule applies within full or block diffusion.

## 4 A CACHEABLE STRUCTURED DECODER FOR DIFFUSION

### 4.1 ONE BACKBONE, ONE RECURSIVE HEAD

In diffusion, the context $C$ includes the noisy sequence $\mathbf{x}_t$ at time $t$, and $\mathbf{y}_A$ is a proposal for the clean tokens $\mathbf{x}_{0,A}$. The bidirectional backbone computes $H_t = (h_v)_{v \in [L]} = \mathrm{Backbone}_\theta(\mathbf{x}_t, t)$. Its native vocabulary layer $G_\theta$ then maps each position independently to logits, $\ell_v = G_\theta(h_v)$. Independent categorical draws give the product law $\mathcal{F}_1$. SPDLM inserts a small conditioning head before this same vocabulary layer and keeps $H_t$ fixed throughout the clean proposal.

The head has parameters $\phi$ and represents each separator by one low-rank key–value slot $\mathcal{C}_{S_v}$. For a position $v$ whose separator is available, READ adds a cache-conditioned residual to $h_v$, which the vocabulary layer maps to logits. For a full-prefix path, WRITE extends the cache using the sampled token's embedding $e(y_v)$:

$$\widetilde{h}_v = \mathrm{Read}_\phi(h_v, \mathcal{C}_{S_v}), \qquad \ell_v = G_\theta(\widetilde{h}_v), \qquad \mathcal{C}_{S_v \oplus v} = \mathrm{Write}_\phi(\widetilde{h}_v, e(y_v)). \tag{5}$$

The cache input determines the retained clean context. For $K = W$, $\widetilde{h}_v$ depends on the preceding prefix, and $S_v \oplus v$ appends the new token. For $K = 2$, we instead write $\mathcal{C}_{\{v\}} = \mathrm{Write}_\phi(h_v, e(y_v))$: the raw backbone state contains no sampled clean ancestors. Writing $\widetilde{h}_v$ would propagate them and violate the one-parent factorization. Both cases share the same operators.

An empty cache gives $\mathrm{Read}_\phi(h, \mathcal{C}_\varnothing) = h$, so $K = 1$ recovers the native head exactly. Figure 3 shows the operators and their execution: positions in each $\mathcal{L}_d$ remain batched, while only the head advances between rounds. The projection and normalization formulas, including the released-model adapters, are given in Sections B.2, D and E. Non-nested separators are rebuilt from their own clean values (Section B.3).

### 4.2 Structured Parallelism and Cost

Let $B$ be the cost of one backbone evaluation, $P$ the native output cost, and $h(|\mathcal{L}_d|)$ the cost of one batched structured-head round. One reverse step costs

$$T_1 = B + P, \qquad T_K = B + H(\mathcal{F}_K), \qquad H(\mathcal{F}_K) = \sum_{d=1}^{D} h(|\mathcal{L}_d|). \tag{6}$$

Topology controls both the number of rounds and their batch sizes. For fixed active sets and topology, the speedup of $N_K$ structured steps over $N_1$ product steps is $(N_1/N_K)/\omega_K$, where $\omega_K = T_K/T_1$ is the measured per-step overhead. At equal quality, fewer backbone calls compensate for the head when $N_1/N_K > \omega_K$. Timings include cache construction and any additional product-logit evaluation; Section B.4 gives the full accounting.

### 4.3 Structured Training and Ancestral Sampling

The normalized forest law is trained by conditional likelihood: with $A = A_n$, $C = C_n$, and gold separator values $\mathbf{x}_{0,S_v}$, the weighted clean-prediction loss is

$$\mathcal{L}_K = \mathbb{E}\left[ -\frac{w(t)}{|A|} \sum_{v \in A} \log p_{\theta,v}(x_0^v \mid \mathbf{x}_{0,S_v}, C) \right]. \tag{7}$$

Here $w(t)$ contains any diffusion-time weighting. Each term is a categorical cross-entropy; empty separators recover the product loss $\mathcal{L}_1$. Clean-token cross-entropy is common in discrete diffusion (Austin et al., 2021; Sahoo et al., 2024; Shi et al., 2024; Gourevitch et al., 2026), but changing the output family does not preserve a tokenwise ELBO derivation automatically. Model-specific weighting, logit conversions, and auxiliary terms are specified in the experimental protocols (Sections C.1, D and E).

At generation time, first sample a clean proposal $\widehat{\mathbf{x}}_{0,A}$ along the forest levels. The known diffusion bridge then samples all coordinates of the next noisy state in parallel, conditional on that proposal. For $s < t$,

$$p_\theta(\mathbf{x}_{s,A} \mid C; \mathcal{F}_K(A)) = \sum_{\widehat{\mathbf{x}}_{0,A}} p_\theta(\widehat{\mathbf{x}}_{0,A} \mid C; \mathcal{F}_K(A)) \prod_{v \in A} q_{s|t}(x_s^v \mid x_t^v, \widehat{x}_0^v). \tag{8}$$

At $K = 1$ this is the product ancestral update; at $K > 1$ the shared proposal can induce dependence in the next state. The forward process, bridge, and active-set schedule are unchanged. Algorithms 1 and 2 detail training and sampling.

## 5 Experiments

We study structured outputs in two settings: pretraining from scratch and adaptation of released diffusion models. TinyStories (Eldan & Li, 2023) makes it possible to vary width, order, topology, and reverse budget under matched training. We also continue pretraining GIDD-3B on Nemotron and adapt Sumi-7B to GSM8K.

The TinyStories backbone is a length-256 DiT with 92.4M non-embedding parameters. TinyStories and Sumi comparisons use three paired training seeds. Following Pynadath et al. (2026), TinyStories

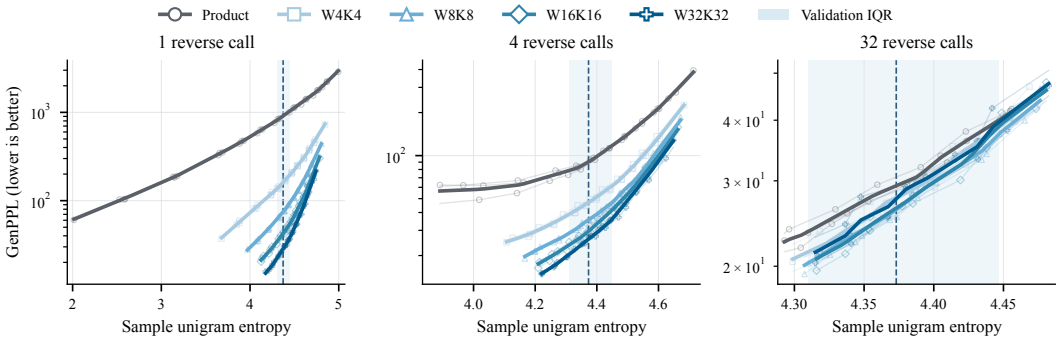



Figure 4: TinyStories entropy–GenPPL frontiers. The blue band is the validation IQR and the dashed line its mean. Dark curves average seeds at matched entropy; faint curves and markers show individual seeds. Width shifts the frontier at low $N$; repeated refinement reduces the gain and produces crossovers.

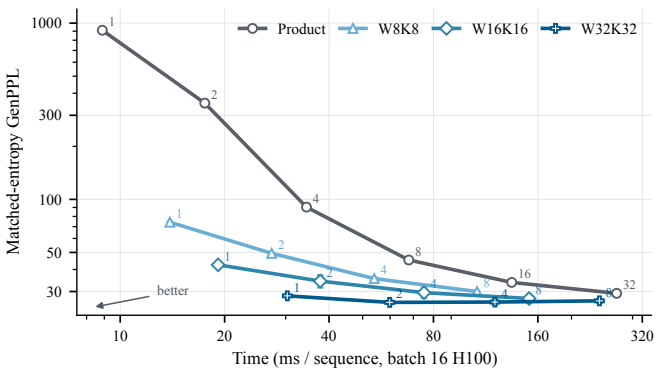
evaluation sweeps temperature to compare sample unigram entropy against generated-text perplexity (GenPPL), scored by GPT-2-large. TinyStories point comparisons use the validation mean entropy, $\bar{H}_{\text{val}} = 4.37$. Timings use isolated H100s; per-sequence times are batch durations divided by batch size, measuring amortized throughput. Full training, aggregation, and timing protocols are in Sections C to E.

### 5.1 TinyStories: Structure and Denoising Steps

We write W$w$K$k$ for component width $W = w$ and conditional order $K = k$; thus W32K32 uses full-prefix conditioning within width-32 components. A trailing -$N$ denotes the number of reverse steps, each requiring one backbone evaluation. Structured models use $\mathcal{L}_K$ with the channel conversion in Section C.1.

**Complete-joint width.** Increasing complete-joint width ($K = W$) shifts the entropy–GenPPL frontier toward lower perplexity at small reverse budgets (Fig. 4). The width advantage persists through $N = 8$ over the supported validation IQR. Curves cross at $N = 16$, and W8K8 outperforms W32K32 at $N = 32$. Local conditioning is most useful when there are few opportunities for global refinement; the complete sweep is in Section C.5.

**Quality and latency.** At the validation mean entropy, W32K32 reaches lower mean GenPPL than Product-32 with one or two backbone calls and substantially less time (Fig. 5). A backbone eval-




| Point | GenPPL | Speedup |
|---|---|---|
| Product-32 | $29.29 \pm 1.08$ | $1.00\times$ |
| W32K32-1 | $28.35 \pm 0.26$ | $8.90\times$ |
| W32K32-2 | $26.03 \pm 0.64$ | $4.51\times$ |
| W32K32-4 | $26.14 \pm 0.81$ | $2.24\times$ |

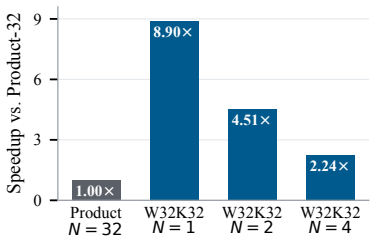



Figure 5: TinyStories at the validation mean entropy. **Left:** controlled batch-16 H100 quality–latency trajectories; labels give $N$ and lower-left is better. **Right:** mean and sample s.d. over three seeds, with measured speedup relative to Product-32.

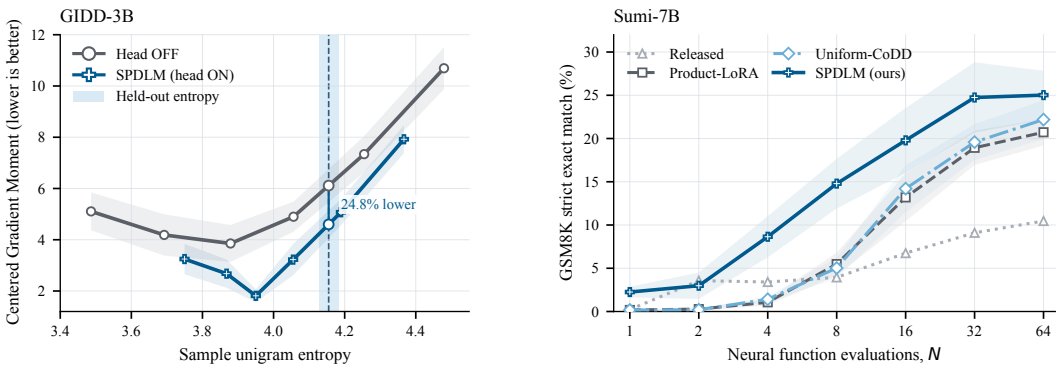



Figure 6: Billion-scale transfer. **Left:** GIDD-3B centered Gradient Moment at eight calls (lower is better), with the same adapted checkpoint's head disabled or enabled. **Right:** full-test Sumi-7B GSM8K accuracy; adapted methods show mean ± s.d. over three seeds.

uation followed by thirty-two local rounds takes 3.41 times a Product step at batch 16, and peak allocation changes from 4.82 to 4.83 GiB.

**Clean context at equal depth.** Do the gains come simply from more sequential rounds? W8K2 and W8K8 both use width-eight components and $D = 8$ left-to-right rounds. The former reads one clean parent; the latter retains the full clean prefix. At the validation mean entropy, W8K8 achieves 17.8–42.8% lower mean GenPPL for $N = 1, 2, 4$, so round count alone does not explain the gain (Fig. 14). Width alone is also insufficient: W16K8 is worse than W8K8, whereas W16K16 is better. The remaining structural controls and same-checkpoint head-OFF evaluation are in Sections C.6 and C.8.

**Comparison with distillation.** Our D-MMD control (Hoogeboom et al., 2026) substantially improves the Product model at $N = 4$, but its frontier remains above non-distilled W32K32 on their shared entropy range (Section C.7).

### 5.2 Transfer to Released Billion-Parameter Diffusion LMs

We use the one-parent W16K2 path for both released models (Fig. 6). GIDD trains an equal-weight mixture of its native product and structured losses to retain the product predictor. Sumi uses its native structured loss alone (Sections D and E).

**GIDD-3B.** On `gidd-unif-3b` (von Rütte et al., 2026), W16K2 lowers centered Gradient Moment (Hoogeboom et al., 2026) from 6.11 to 4.60 at the held-out entropy 4.16, a 24.8% reduction using interpolation within measured support. The measured discrepancy is lower at every tested temperature. Both paths use the same adapted checkpoint, with the structured head disabled or enabled. The metric compares generated and held-out continuations in reference-model gradient features; lower is better. The head's H100 overhead falls from 12.65% at batch one to 2.33% at batch eight (Section D).

**Sumi-7B.** Starting from Sumi-7B (Ye et al., 2026), we train rank-16 Product-LoRA and W16K2 SPDLM adapters for one GSM8K epoch (Cobbe et al., 2021). Our uniform-noise port of CoDD (Li et al., 2026) provides a second adapted baseline: a 1,024-state probabilistic circuit recouples the Transformer's $\mathcal{F}_1$ potentials into a dependent joint law, while SPDLM directly parameterizes clean-token conditionals. Product-LoRA and SPDLM share update budgets; Uniform-CoDD trains its circuit for one additional epoch. We evaluate all 1,319 test questions at $N = 1, 2, 4, 8, 16, 32, 64$, using the same prompt, bridge, answer window, and reverse budget (Section E).

SPDLM has higher mean accuracy than both adapted baselines at every tested budget (Table 1). At $N = 1$, accuracy is still low, but reaches 2.25% versus 0.18% for Product-LoRA and Uniform-CoDD. The latter coincide because Uniform-CoDD activates its circuit only at lower noise levels,

Table 1: Full-test GSM8K strict exact match (%). Adapted methods report mean ± sample s.d. over three training seeds; Released denotes the original checkpoint with the same ancestral sampler. Best mean accuracy in bold.

| $N$ | Released | Product-LoRA | Uniform-CoDD | SPDLM (ours) |
|---|---|---|---|---|
| 1 | 0.23 | $0.18 \pm 0.04$ | $0.18 \pm 0.04$ | $\mathbf{2.25 \pm 0.62}$ |
| 2 | **3.56** | $0.28 \pm 0.22$ | $0.20 \pm 0.09$ | $2.98 \pm 1.54$ |
| 4 | 3.41 | $1.06 \pm 0.35$ | $1.42 \pm 0.35$ | $\mathbf{8.64 \pm 2.39}$ |
| 8 | 3.94 | $5.48 \pm 1.33$ | $5.03 \pm 1.21$ | $\mathbf{14.78 \pm 2.88}$ |
| 16 | 6.75 | $13.19 \pm 2.76$ | $14.20 \pm 2.72$ | $\mathbf{19.79 \pm 3.75}$ |
| 32 | 9.10 | $18.93 \pm 1.99$ | $19.59 \pm 2.08$ | $\mathbf{24.74 \pm 4.07}$ |
| 64 | 10.46 | $20.72 \pm 1.54$ | $22.19 \pm 2.24$ | $\mathbf{25.02 \pm 2.83}$ |

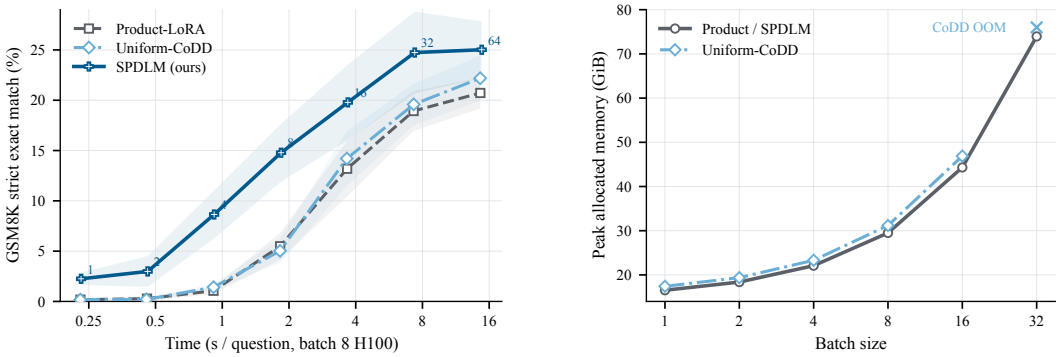



Figure 7: Sumi-7B systems results on one H100. **Left:** strict GSM8K accuracy against measured batch-8 latency; SPDLM points are labeled by $N$. **Right:** peak memory at $N = 32$; Product and SPDLM coincide, while Uniform-CoDD is out of memory at batch 32.

so its one-step evaluation uses the product path (Section E). At $N = 32$, SPDLM reaches $24.74\%$ versus $18.93\%$ and $19.59\%$, and exceeds Uniform-CoDD-64 with half as many calls. Each of the three training seeds favors SPDLM over both adapted baselines at all seven budgets (Section E).

### 5.3 Task Accuracy and Systems Cost at 7B Scale

At batch eight, SPDLM-32 takes $7.38$ seconds per question, roughly half the time of Product-LoRA-64 and Uniform-CoDD-64 (Fig. 7). Against Product-LoRA at the same $N = 32$, median time rises by $1.23\%$ and measured peak allocation is unchanged. Our Uniform-CoDD implementation uses $5.4$–$5.8\%$ more memory through batch 16 and exceeds the 80-GiB H100 capacity at batch 32.

## 6 Conclusion

SPDLMs model dependencies among tokens proposed in the same denoising step. A lightweight decoder uses the representations from one Transformer pass to condition token predictions on previously sampled tokens. This reduces the need for later denoising steps to coordinate these predictions. On TinyStories, SPDLMs improve the trade-off between generative perplexity and latency at matched unigram entropy, with the largest gains when generation uses few steps. Adaptation experiments on GIDD-3B and Sumi-7B also show benefits with little additional runtime or memory cost.

**Limitations.** Compressing the clean separator into one state can limit prediction accuracy; other caches and read–write rules may improve the trade-off. We use fixed partitions and topologies. Full pretraining is limited to TinyStories; billion-scale results concern adaptation of released checkpoints. All experiments use uniform corruption, and large-scale reasoning is evaluated only on GSM8K. Matching unigram entropy in TinyStories leaves higher-order diversity unconstrained.

## AI USE STATEMENT

We used an AI agent for implementation, checking mathematical correctness, literature checks, and figures. The authors are responsible for the paper's claims, results, and code.

## REFERENCES


Marianne Arriola and Volodymyr Kuleshov. Set diffusion: Interpolating token orderings between autoregression and diffusion for fast and flexible decoding. *arXiv preprint arXiv:2607.01775*, 2026.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar 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. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/7ede97c3e082c6df10a8d6103a2eebd2-Paper-Conference.pdf.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. Flow map matching with stochastic interpolants: A mathematical framework for consistency models. *arXiv preprint arXiv:2406.07507*, 2024.

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.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968. doi: 10.1109/TIT.1968.1054142. URL https://doi.org/10.1109/TIT.1968.1054142.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Justin Deschenaux and Caglar Gulcehre. BlockGen: Flexible blockwise sequence modeling with hybrid samplers. *arXiv preprint arXiv:2606.02241*, 2026.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, Tianlin Liu, Ivan Lobov, Ofir Nabati, João Gabriel Oliveira, Nicolas Perez-Nieves, Nastasia Prutianova, Bobak Shahriari, Jean Tarbouriech, Pavel Tyletski, Çağlar Ünlü, Cindy Wu, Glenn Cameron, Jerome Connor, Sertan Girgin, Maarten Grootendorst, Alon Levkovitch, Eliya Nachmani, Omar Sanseviero, Piotr Stanczyk, Quentin Berthet, Andrew Campbell, Clément Crépy, Valentin De Bortoli, Arnaud Doucet, Romuald Elie, Alexandre Galashov, Klaus Greff, Alexis Jacq, David Ruhe, Yu-Han Wu, Sebastian Flennerhag, Brendan O'Donoghue, George Scrivener, and Shantanu Thakoor. DiffusionGemma technical report. *arXiv preprint arXiv:2608.00146*, 2026.

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Samson Gourevitch, Yazid Janati, Dario Shariatian, Umut Simsekli, Eric Moulines, Eric P. Xing, and Alain Durmus. Uniform diffusion models revisited: Leave-one-out denoiser and absorbing state reformulation. *arXiv preprint arXiv:2605.22765*, 2026.

Michael Hersche, Nicolas Menet, Ronan Tanios, and Abbas Rahimi. Locally coherent parallel decoding in diffusion language models. *arXiv preprint arXiv:2603.20216*, 2026.

Emiel Hoogeboom, David Ruhe, Jonathan Heek, Thomas Mensink, and Tim Salimans. Beyond single tokens: Distilling discrete diffusion models via discrete MMD. *arXiv preprint arXiv:2603.20155*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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.

Edith Kovács and Tamás Szántai. On the approximation of a discrete multivariate probability distribution using the new concept of t-Cherry junction tree. In *Coping with Uncertainty: Robust Solutions*, volume 633 of *Lecture Notes in Economics and Mathematical Systems*, pp. 39–56. Springer Berlin Heidelberg, October 2009. ISBN 9783642037351. doi: 10.1007/978-3-642-03735-1_3. URL https://doi.org/10.1007/978-3-642-03735-1_3.

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.

Ian Li, Zilei Shao, Benjie Wang, Rose Yu, Guy Van den Broeck, and Anji Liu. Breaking the factorization barrier in diffusion language models. *arXiv preprint arXiv:2603.00045*, 2026.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 4328–4343. Curran Associates, Inc., 2022. doi: 10.52202/068431-0313. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/1be5bc25d50895ee656b8c2d9eb89d6a-Paper-Conference.pdf.

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, Main Conference, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

Shen Nie, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. *arXiv preprint arXiv:2606.25331*, 2026.

Patrick Pynadath, Jiaxin Shi, and Ruqi Zhang. Generative frontiers: Why evaluation matters for diffusion language models. *arXiv preprint arXiv:2604.02718*, 2026.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub Tomczak, and Cheng

Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167. Curran Associates, Inc., 2024. doi: 10.52202/079017-3277. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bad233b9849f019aead5e5cc60cef70f-Paper-Conference.pdf.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. URL https://arxiv.org/abs/1706.03762.

Dimitri von Rütte, Janis Fluri, Omead Pooladzandi, Bernhard Schölkopf, Thomas Hofmann, and Antonio Orvieto. Scaling behavior of discrete diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 36638–36663, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/3dd84aeae46ad92cdff18b3c9d68ab21-Paper-Conference.pdf.

Mengyu Ye, Keito Kudo, Wataru Ikeda, Ryosuke Matsuda, Keisuke Sakaguchi, and Jun Suzuki. Sumi: Open uniform diffusion language model from scratch. *arXiv preprint arXiv:2606.19005*, 2026.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, et al. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, 2026.

# A Additional Theory

## A.1 Optimal Product Approximation

Fix an active set $A$ and write $\mathbf{y}_A = (y_v)_{v\in A}$. For completeness, define the conditional total correlation of the values proposed in one update as $\mathrm{TC}_{p^*}(\mathbf{y}_A \mid C) = \sum_{v\in A} H_{p^*}(y_v \mid C) - H_{p^*}(\mathbf{y}_A \mid C)$.

**Proposition A.1** (Best possible product predictor)**.** *For any target conditional $p^*(\mathbf{y}_A \mid C)$,*

$$\min_{p\in\mathcal{P}_{\mathrm{prod}}} \mathbb{E}_{p^*}[-\log p(\mathbf{y}_A \mid C)] = H_{p^*}(\mathbf{y}_A \mid C) + \mathrm{TC}_{p^*}(\mathbf{y}_A \mid C). \tag{9}$$

*Here $\mathcal{P}_{\mathrm{prod}}$ is the family of product conditionals. The optimum uses the exact marginals $p_v(\cdot \mid C) = p^*(y_v \mid C)$.*

*Proof.* For any product conditional $p(\mathbf{y}_A \mid C) = \prod_{v\in A} p_v(y_v \mid C)$, the cross-entropy decomposes as $\mathbb{E}_{p^*}[-\log p(\mathbf{y}_A \mid C)] = \sum_{v\in A} H_{p^*}(y_v \mid C) + \sum_{v\in A} \mathbb{E}_C\,\mathrm{KL}(p^*(y_v \mid C)\|p_v(y_v \mid C))$. The KL terms are minimized at $p_v = p^*(y_v \mid C)$. The remaining gap to $H_{p^*}(\mathbf{y}_A \mid C)$ is precisely $\mathrm{TC}_{p^*}(\mathbf{y}_A \mid C)$, proving the result. □

## A.2 The Fixed-Separator Gain

We write $\mathcal{R}$ for the unweighted, summed conditional cross-entropy in these approximation results. It is distinct from the time-weighted, per-token training loss $\mathcal{L}_K$ in Eq. (7).

**Proposition A.2** (General fixed-separator gain). *For any $K$, let $\mathsf{S} = (S_v)_{v\in T}$ be a fixed valid separator family. If every factor can represent its population conditional, then*

$$\mathcal{R}^*_{\mathrm{prod}} - \mathcal{R}^*_{\mathsf{S}} = \sum_{v\in T} I_{p^*}(y_v; \mathbf{y}_{S_v} \mid C), \tag{10}$$

*where the contribution is zero when $S_v = \varnothing$.*

*Proof.* For a fixed component $T$ and valid separator family $\mathsf{S} = (S_v)_{v\in T}$, the optimal factor conditionals are the corresponding data conditionals, with minimum cross-entropy $\mathcal{R}^*_{\mathsf{S}} = \sum_{v\in T} H(y_v \mid \mathbf{y}_{S_v}, C)$. Since the product optimum is $\sum_{v\in T} H(y_v \mid C)$, their difference is $\mathcal{R}^*_{\mathrm{prod}} - \mathcal{R}^*_{\mathsf{S}} = \sum_{v\in T} I(y_v; \mathbf{y}_{S_v} \mid C)$, proving Proposition A.2. For an order-2 tree, $S_{v_\circ} = \varnothing$ and $S_v = \{\mathrm{pa}(v)\}$ otherwise, which gives the stated sum over tree edges. Summing the identity over disjoint components gives the corresponding fixed-forest result. □

For an order-two tree, the gain is $\mathcal{R}^*_{\mathrm{prod}} - \mathcal{R}^*_{\mathsf{S}} = \sum_{v\in T\setminus\{v_\circ\}} I(y_v; y_{\mathrm{pa}(v)} \mid C)$: each edge recovers conditional dependence discarded by the product law. More generally, adding a clean variable $u$ to a separator $S_v$ has ideal incremental gain $H(y_v \mid C, \mathbf{y}_{S_v}) - H(y_v \mid C, \mathbf{y}_{S_v}, y_u) = I(y_v; y_u \mid C, \mathbf{y}_{S_v})$.

### A.3 Relationship to the KL-Optimal Tree

Let $p^*(\mathbf{y}_T \mid C)$ be fixed and let $p^*_{\mathcal{F}}$ use exact root and edge conditionals. Specializing Proposition A.2 to order 2 and combining it with the product-gap result gives

$$\mathbb{E}_C \,\mathrm{KL}(p^*(\mathbf{y}_T \mid C) \| p^*_{\mathcal{F}}(\mathbf{y}_T \mid C)) = \mathrm{TC}(\mathbf{y}_T \mid C) - \sum_{(u\to v)\in E} I(y_v; y_u \mid C). \tag{11}$$

For one connected tree, minimizing Eq. (11) amounts to the Chow–Liu maximum-weight spanning-tree problem with conditional mutual-information weights (Chow & Liu, 1968). We instead fix the topology and train its normalized conditionals. For a fixed undirected tree, changing the root leaves the population-optimal distribution unchanged, but changes the neural parameterization and sampling depth.

### A.4 The Joint Approximation Hierarchy

The *order* is the maximum clean-variable clique size. Consider one component $T$ of width $W$ with construction order $\pi = (v_1, \ldots, v_W)$. The product law is

$$p_\theta(\mathbf{y}_T \mid C; \mathcal{F}_1(T)) = \prod_{v\in T} p_{\theta,v}(y_v \mid C) = \prod_{r=1}^{W} p_{\theta,v_r}(y_{v_r} \mid C). \tag{12}$$

Ordering these factors alone introduces no clean dependence. An order-$K$ t-cherry junction tree starts from $\Gamma_0 = \{v_1, \ldots, v_K\}$ and adds each remaining position $v_r$ through a separator $S_{v_r}$ of size $K-1$ contained in an existing clique (Kovács & Szántai, 2009):

$$p_K(\mathbf{y}_T \mid C) = p_{\Gamma_0}(\mathbf{y}_{\Gamma_0} \mid C) \prod_{r=K+1}^{W} p_{v_r}(y_{v_r} \mid \mathbf{y}_{S_{v_r}}, C), \qquad |S_{v_r}| = K-1. \tag{13}$$

The new clique is $S_{v_r} \cup \{v_r\}$. Requiring the separator to lie in an existing clique gives the junction-tree construction; a cardinality bound and an acyclic order alone would also allow graphs outside this family. Expanding the seed-clique joint by the chain rule yields the tokenwise form in Eq. (3), with shorter separators for the first $K$ variables. The initial clique is therefore not a separate categorical

table in our decoder: its variables are generated in successive head rounds. The endpoints are

$$\begin{cases} K=1: & \displaystyle\prod_{v\in T} p_v(y_v \mid C) & \text{(product)},\\ K=2: & p_{v_\circ}(y_{v_\circ}\mid C)\displaystyle\prod_{v\in T\setminus\{v_\circ\}} p_v(y_v \mid y_{\mathrm{pa}(v)}, C) & \text{(tree)},\\ & \vdots & \\ K=W: & p_W(\mathbf{y}_T \mid C) & \text{(unrestricted joint)}. \end{cases} \tag{14}$$

For a fixed structure $\mathsf{S}$ with population-optimal factors, $\mathcal{R}^*_{\mathsf{S}} = H(\mathbf{y}_{\Gamma_0} \mid C) + \sum_{r=K+1}^{W} H(y_{v_r} \mid \mathbf{y}_{S_{v_r}}, C)$, and its gain over product is $\mathcal{R}^*_{\mathrm{prod}} - \mathcal{R}^*_{\mathsf{S}} = \mathrm{TC}(\mathbf{y}_{\Gamma_0} \mid C) + \sum_{r=K+1}^{W} I(y_{v_r}; \mathbf{y}_{S_{v_r}} \mid C)$. Let $\mathcal{P}_K^{(W)}$ collect the laws with unrestricted conditional factors over these structures of order at most $K$, including their lower-order members. Then $\mathcal{P}_1^{(W)} \subseteq \cdots \subseteq \mathcal{P}_W^{(W)}$, and their optimal risks satisfy $\mathcal{R}^*_1 \geq \cdots \geq \mathcal{R}^*_W$, where $\mathcal{R}^*_K = \inf_{q\in\mathcal{P}_K^{(W)}} \mathbb{E}[-\log q(\mathbf{y}_T \mid C)]$. At $K = W$ the graphical family is unrestricted; lower orders suffice when the target already admits the corresponding factorization. A trained neural head may not attain these optima or preserve their ordering.

Order does not define an architecture. At $K = W$, an explicit categorical table realizes the unrestricted joint in one sampling operation but requires $O(|\mathcal{V}|^W)$ probabilities. Writing $V_{<r} = \{v_1, \ldots, v_{r-1}\}$, the chain rule instead realizes the same joint as

$$p_W(\mathbf{y}_T \mid C) = \prod_{r=1}^{W} p_{v_r}(y_{v_r} \mid \mathbf{y}_{V_{<r}}, C), \tag{15}$$

which avoids materializing the table but samples sequentially. A recursive one-slot realization replaces the raw prefix by a deterministic learned state $s_r = U_\phi(s_{r-1}, h_{v_r}, e(y_{v_r}))$ and remains a normalized chain-rule joint, but compression may restrict its conditionals. Graphical order and neural expressivity are separate properties.

### A.5 Component Width and Sampling Depth

For a fixed structure, assign a position its earliest sampling round $d_v = 1 + \max_{u\in S_v} d_u$, with $\max \varnothing = 0$. A component has depth $D_m = \max_{v\in T_m} d_v$; a forest has $D = \max_m D_m$ and parallel levels $\mathcal{L}_d = \{v : d_v = d\}$. Grouping its normalized factors by components or by levels gives the same law:

$$p_\theta(\mathbf{y}_{A_n} \mid C_n; \mathcal{F}_K(A_n)) = \prod_{m=1}^{M} \prod_{v\in T_m} p_{\theta,v}(y_v \mid \mathbf{y}_{S_v}, C_n) = \prod_{d=1}^{D} \prod_{v\in\mathcal{L}_d} p_{\theta,v}(y_v \mid \mathbf{y}_{S_v}, C_n). \tag{16}$$

Components may have unequal widths $W_m = |T_m|$, in which case $\sum_m W_m = |A_n|$ and $W = \max_m W_m$. When all components have width $W$ and $W$ divides $|A_n|$, $MW = |A_n|$; otherwise the final component is shorter.

Figure 8 holds $W = 6$ and $K = 2$ fixed and varies only the topology. A star has depth two, a balanced tree has logarithmic depth, and a chain such as the left-to-right path has $D = W$. Neither width nor conditional order alone determines the number of sequential rounds.

## B Decoder and Algorithmic Details

### B.1 Model Configurations

The TinyStories model used in the main comparison has $(W, K, D) = (32, 32, 32)$ and uses the pure objective $\mathcal{L}_K$. The GIDD-3B experiment in Section 5.2 uses the $(16, 2, 16)$ path and $\mathcal{J}_{\mathrm{mix}}$, the equal-weight native-surrogate objective defined in Section D. Sumi-7B uses the same $(16, 2, 16)$ path with its pure native structured objective (Section E).

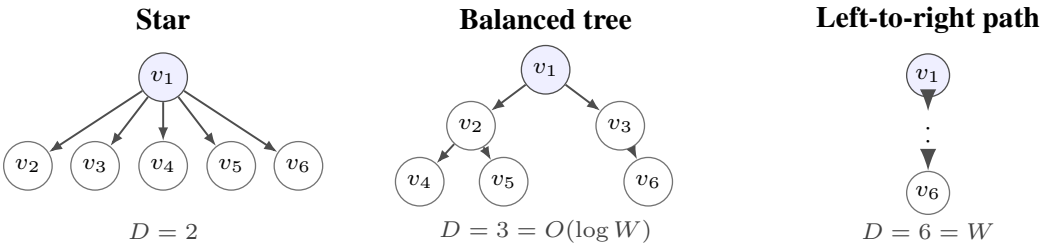



Figure 8: Sampling depth is a separate axis. Holding $W = 6, K = 2$ fixed, topology changes $D$; nodes at the same level are parallel.

### B.2 Read and Write Operators

The one-slot head in Fig. 3 combines a contextual feature with the clean token embedding, then projects the result to a key and value. For a nonempty cache, the TinyStories implementation is

$$\begin{aligned} m_v &= 2^{-1/2}\Big[\mathrm{LN}_{\mathrm{ctx}}(\widetilde{h}_v) + \mathrm{LN}_{\mathrm{tok}}(e(y_v))\Big], \\ \mathrm{Write}_\phi(\widetilde{h}_v, e(y_v)) &= (W_k m_v, W_v m_v), \\ \mathrm{Read}_\phi(h, \mathcal{C}) &= h + W_o W_a \,\mathrm{Attn}(W_q \,\mathrm{LN}_{\mathrm{child}}(h); \{\varnothing, \mathcal{C}\})\,. \end{aligned} \tag{17}$$

Here Attn is multi-head attention over the separator slot and a learned null key with zero value, so the read can suppress an unhelpful cache. An empty separator bypasses the correction. The output projection $W_o$ is zero-initialized, and there is no additional conditional FFN. The residual precedes the native final normalization. Released-model adapters retain the same cache interface but add a query residual and use model-specific output normalization (Sections D and E).

### B.3 Separator Cache Construction

A cache is keyed by its ordered clean separator, not merely by the last position visited. To construct $\mathcal{C}_S$ for $S = (s_1, \ldots, s_p)$, start from the empty cache and successively compute $\widetilde{h}_{s_j} = \mathrm{Read}_\phi(h_{s_j}, \mathcal{C}_{(s_1,\ldots,s_{j-1})})$ and $\mathcal{C}_{(s_1,\ldots,s_j)} = \mathrm{Write}_\phi(\widetilde{h}_{s_j}, e(y_{s_j}))$. Reuse a cached prefix only if its ordered separator matches. For a singleton, this reduces to $\mathrm{Write}_\phi(h_{s_1}, e(y_{s_1}))$. For a non-nested separator, rebuild from its own clean values: carrying a previously conditioned state would introduce undeclared ancestors and change the factorization. The algorithms below use this construction with gold values during training and generated values during sampling.

### B.4 Detailed Cost Accounting

The compact cost model in Eq. (6) hides implementation-level batch sizes. For reverse step $n$, let $a_n = |A_n|$, $b_{n,d} = |\mathcal{L}_{n,d}|$, and let $u_{n,d} \le b_{n,d}$ count positions whose outputs extend a cache. Because the levels partition the active set, $\sum_d b_{n,d} = a_n$. Writing $\tau_{\mathrm{bb}}$ for one backbone call and $\tau_{\mathrm{read}}$, $\tau_{\mathrm{vocab}}$, $\tau_{\mathrm{sample}}$, and $\tau_{\mathrm{write}}$ for batched head kernels gives

$$T_{1,n} = \tau_{\mathrm{bb}}(L) + \tau_{\mathrm{vocab}}(a_n) + \tau_{\mathrm{sample}}(a_n), \tag{18}$$

$$T_{K,n} = \tau_{\mathrm{bb}}(L) + \sum_{d=1}^{D_n} [\tau_{\mathrm{read}}(b_{n,d}) + \tau_{\mathrm{vocab}}(b_{n,d}) + \tau_{\mathrm{sample}}(b_{n,d}) + \tau_{\mathrm{write}}(u_{n,d})]\,. \tag{19}$$

This accounting describes a fused execution in which every active position is projected and sampled once. Non-nested cache reconstruction adds read/write work; a wrapper that also computes native product logits adds projection work. Measured timings include these implementation costs. For nested prefixes or raw singleton caches, the extra read/write work for head rank $r_{\mathrm{head}}$ and backbone width $d_{\mathrm{model}}$ is $O((a_n + \sum_d u_{n,d}) d_{\mathrm{model}} r_{\mathrm{head}})$; $D_n$ mainly controls kernel serialization and the per-level batch sizes. For a regular partition with $a_n = MW$, a width-$W$ chain has $D_n = W$ and $b_{n,d} = M$. Other topologies can hold $(M, W, K)$ fixed while reducing $D_n$ by exposing more

positions in each level. Thus $W$ controls local coverage, $K$ controls conditional order, $D_n$ controls the head critical path, and $M$ controls component-level parallelism.

For fixed active sets and topology, let $\omega_K = T_K/T_1$. Comparing $N_1$ product calls with $N_K$ structured calls gives

$$\text{Speedup} = \frac{N_1 T_1}{N_K T_K} = \frac{N_1/N_K}{\omega_K}. \tag{20}$$

At matched quality, the structured sampler is faster when $N_1/N_K > \omega_K$.

### B.5 Training and Sampling

**Algorithm 1** Pure or mixed order-$K$ diffusion training

**Require:** Clean sequence $\mathbf{x}_0$, active set $A$, forest $\mathcal{F}_K(A)$ with levels $\mathcal{L}_1, \dots, \mathcal{L}_D$, objective $\mathcal{L}_K$ or $\mathcal{L}_{\text{mix}}$

Sample diffusion time(s) and noisy state $\mathbf{x}_t$
Compute shared contextual features $H_t = \text{Backbone}_\theta(\mathbf{x}_t, t)$
Initialize the empty cache $\mathcal{C}_\varnothing$ for every component
**for** $d = 1$ to $D$ **do**
  **for** all $v \in \mathcal{L}_d$ in parallel **do**
    $\widetilde{h}_v \leftarrow \text{Read}_\phi(h_v, \mathcal{C}_{S_v})$
    $\ell_v \leftarrow G_\theta(\widetilde{h}_v)$ and accumulate $-\log p_{\theta,v}(x_0^v \mid \mathbf{x}_{0,S_v}, C)$
  **end for**
  **for** all newly available separators $S$ needed by a later factor **do**
    Construct or reuse $\mathcal{C}_S$ from $\mathbf{x}_{0,S}$ as in Section B.3
  **end for**
**end for**
Form $\mathcal{L}_K$ from the accumulated structured cross-entropies
**if** using $\mathcal{L}_{\text{mix}}$ **then**
  Evaluate $G_\theta(h_v)$ for all $v \in A$ and form $\mathcal{L}_1$
  Set $\mathcal{L}_{\text{mix}} \leftarrow \frac{1}{2}(\mathcal{L}_1 + \mathcal{L}_K)$
**end if**
Update $(\theta, \phi)$ using the selected objective

The displayed cross-entropy is the generic clean-prediction objective. When a model uses a channel-specific logit conversion, it is applied to $\ell_v$ before the target cross-entropy. Section C.1 gives the TinyStories conversion. GIDD and Sumi instead evaluate their native surrogate on these logits, as specified in their adaptation protocols.

**Algorithm 2** Bounded-depth structured reverse update

**Require:** Current state $\mathbf{x}_t$, earlier time $s < t$, active set $A$, forest $\mathcal{F}_K(A)$ with levels $\mathcal{L}_1, \dots, \mathcal{L}_D$

Compute shared features $H_t = \text{Backbone}_\theta(\mathbf{x}_t, t)$ once
Initialize the empty cache $\mathcal{C}_\varnothing$ for every component
**for** $d = 1$ to $D$ **do**
  **for** all $v \in \mathcal{L}_d$ in parallel **do**
    $\widetilde{h}_v \leftarrow \text{Read}_\phi(h_v, \mathcal{C}_{S_v})$
    $\ell_v \leftarrow G_\theta(\widetilde{h}_v)$ and $\widehat{x}_0^v \sim p_{\theta,v}(\cdot \mid \widehat{\mathbf{x}}_{0,S_v}, C)$
  **end for**
  **for** all newly available separators $S$ needed by a later factor **do**
    Construct or reuse $\mathcal{C}_S$ from $\widehat{\mathbf{x}}_{0,S}$ as in Section B.3
  **end for**
**end for**
For every $v \in A$, sample $x_s^v \sim q_{s|t}(\cdot \mid x_t^v, \widehat{x}_0^v)$ in parallel
Return the complete earlier-time state $\mathbf{x}_s$

| Setting | TinyStories pretraining |
|---|---|
| Tokenization | GPT-2 tokenizer; EOS-delimited text packed into 254-token chunks, with BOS and EOS added to each chunk |
| Backbone | 12 DiT blocks, hidden width 768, 12 attention heads, dropout 0.1; untied input and output embeddings |
| Batch and precision | 48 sequences per update on one H100; bfloat16 |
| Optimizer | AdamW, learning rate $3.5 \times 10^{-4}$, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.01 |
| Schedule | 500-update linear warmup, then constant learning rate; gradient clipping at 1 |
| Training and selection | 25,000 updates; validation and checkpointing every 5,000 updates; evaluate the final checkpoint; no EMA |

Table 2: Shared TinyStories training settings. Only the structured head and its configuration change between the compared models.

# C TinyStories: Protocols and Complete Results

## C.1 TinyStories Training Protocol

All main TinyStories runs use length $L = 256$, one full diffusion block, a DiT with 92.4M non-embedding parameters, uniform corruption, and 25,000 optimizer updates without EMA. Generation uses the ancestral sampler in Eq. (8). The parameter count excludes the input embedding and untied vocabulary projection. Product and structured arms share the tokenizer, data stream, batch size, warmup, optimizer, precision, and checkpoint schedule in Table 2. We use three paired training seeds. Structured arms use $\mathcal{L}_K$ alone.

This implementation uses the leave-one-out conversion of Gourevitch et al. (2026). The uniform forward channel is $q_t(x_t^v = b \mid x_0^v = a) = \alpha_t \mathbf{1}\{b = a\} + (1 - \alpha_t)\rho(b)$, with $\rho(b) = |\mathcal{V}|^{-1}$. If $\nu_{\theta,v}(a \mid \mathbf{x}_{0,S_v}, C)$ denotes the structured head's clean evidence, the categorical posterior used in the cross-entropy of Eq. (7) is

$$\mu^{(K)}_{\theta,v}(a \mid \mathbf{x}_{0,S_v}, C) = \frac{q_t(x_t^v \mid a)\nu_{\theta,v}(a \mid \mathbf{x}_{0,S_v}, C)}{\sum_{b \in \mathcal{V}} q_t(x_t^v \mid b)\nu_{\theta,v}(b \mid \mathbf{x}_{0,S_v}, C)}. \tag{21}$$

The active set is partitioned into contiguous width-$W$ components, with the final component truncated when necessary. Here $W$ is the maximum width of one local joint, not the global diffusion block size, and $M = \lceil |A|/W \rceil$ components run concurrently. The full-prefix family has $K = D = W$ and follows the local chain rule. TinyStories holds the initial BOS token fixed and excludes it from the loss. A length-256 W32K32 update therefore has 255 active positions: seven width-32 components and one width-31 component, with at most 32 head rounds across components.

Every structured model uses a one-slot merged-attention head in hidden residual space, with rank 256 and eight heads. Caches encode the declared separator: singleton caches use raw parent features, while nested prefixes compose conditioned features and clean token embeddings as in Section B.3. No parameter depends on $K$; an empty separator at $K = 1$ recovers the classical product vocabulary head.

## C.2 Frontier and Aggregation Protocol

We compute generated-text perplexity (GenPPL) with GPT-2-large. Sample entropy is the mean per-sequence unigram entropy of the GPT-2 token IDs, computed in nats over the full length-256 samples.

We sweep temperatures 0.45, 0.55, 0.65, 0.75, 0.80, 0.85, 0.90, 0.95, 1.00, 1.05, 1.10, and 1.20 for $N \in \{1, 2, 4\}$. For $N \in \{8, 16, 32\}$, we use 0.75–1.10 in increments of 0.05. Each point contains 256 samples at batch size 16, with a shared evaluation seed. For each model, training seed, and $N$, we sort the measured points by entropy and linearly interpolate log GenPPL between neighboring points. At each entropy, we convert back to GenPPL and take the arithmetic mean across seeds.

Each mean curve is restricted to the entropy range covered by all three seeds; comparisons between models use the overlap of their ranges. We never extrapolate. Faint curves and markers show individual seeds and measured points within that shared range.

All TinyStories point comparisons use the empirical validation mean entropy, $\bar{H}_{\text{val}} = 4.37$, computed at full precision before interpolation. This includes the quality–latency table, structural controls, distillation, and same-checkpoint attribution. Each uses the 256-sample temperature sweeps described above. The full frontier curves show how the comparisons change away from this target. Error bars show the sample standard deviation across training seeds.

### C.3 Validation Entropy Distribution

We compute the operating region from all 21,508 TinyStories validation sequences, with the same tokenizer, length-256 wrapping, EOS handling, and entropy calculation used for generated text. The resulting distribution has mean 4.37 (s.d. 0.11). We use this mean as the comparison target and shade the interquartile region to show the spread of validation entropies (Fig. 9).

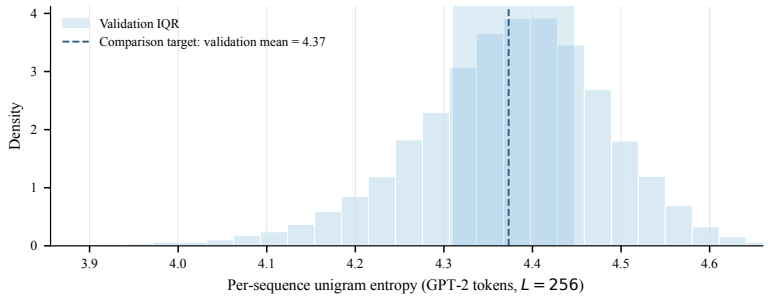



Figure 9: TinyStories validation entropy distribution under the frontier tokenizer and sequence construction. The dashed line marks the mean used for point comparisons; the blue band shows the interquartile region.

### C.4 Controlled Latency Protocol

The latency benchmark covers Product, W8K2, W8K8, W16K8, W16K16, and W32K32 at batch sizes 1, 8, and 16. Its 96 cells run sequentially in one H100 process with identical precision and sampler settings, 10 warmups, 50 synchronized measurements, and no concurrent jobs. At batch 16, W8K8, W16K16, and W32K32 cost $1.57\times$, $2.16\times$, and $3.41\times$ a Product call. The largest peak-allocation difference is 0.01 GiB.

### C.5 All Reverse Budgets

Figure 10 extends the main frontier comparison to all six reverse budgets.

At the validation mean entropy, increasing complete-joint width improves GenPPL through $N = 16$, with progressively smaller gains. At $N = 32$, W8K8 and W16K16 outperform W32K32. Validation structured NLL becomes less predictive of generated-sample GenPPL as $N$ grows: Spearman correlation falls from 1.00 at $N = 1$ to 0.69 at $N = 32$. Both values use all 17 designs, with three-seed entropy support at every budget, correlating mean validation NLL with mean matched-entropy GenPPL.

Matched comparisons use the intersection of the three measured seed ranges shown in Fig. 11.

### C.6 Controlled Order, Coverage, Depth, and Topology

The order and coverage studies fix component width but can also change depth (Figs. 12 and 13). The equal-depth comparison fixes both $W = D = 8$ and changes only the available clean context (Fig. 14). The topology study fixes $W = 8, K = 2$ and varies the tree, which also changes its sampling depth (Fig. 15).

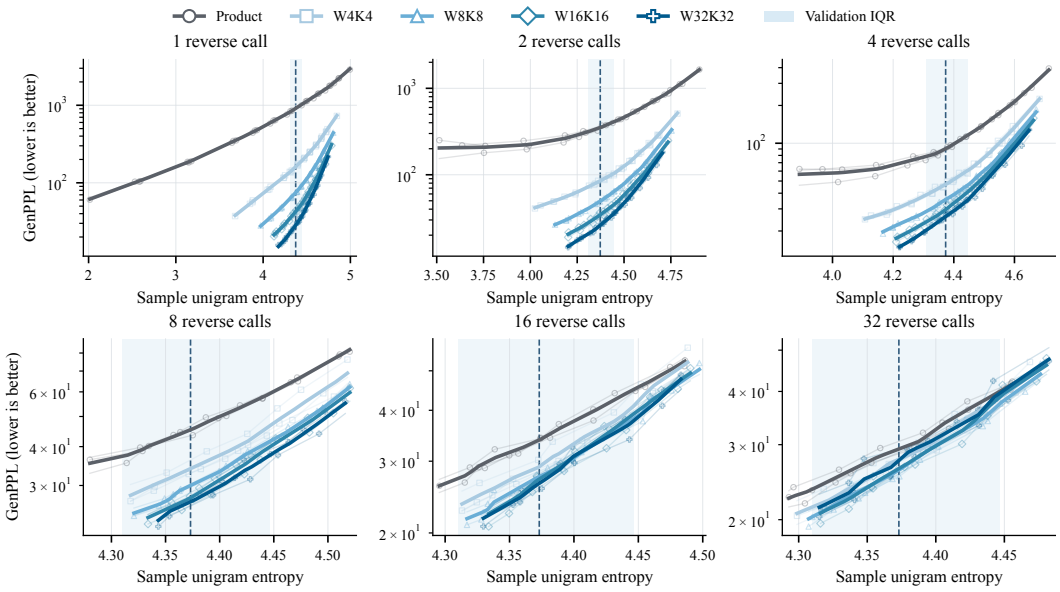



Figure 10: Entropy–GenPPL frontiers for $N \in \{1, 2, 4, 8, 16, 32\}$. Dark curves average seeds at matched entropy; faint curves and markers show individual seeds. The blue band and dashed line denote the validation IQR and mean. Higher-budget panels use tighter entropy windows. Each mean curve stops at the boundary of three-seed support.

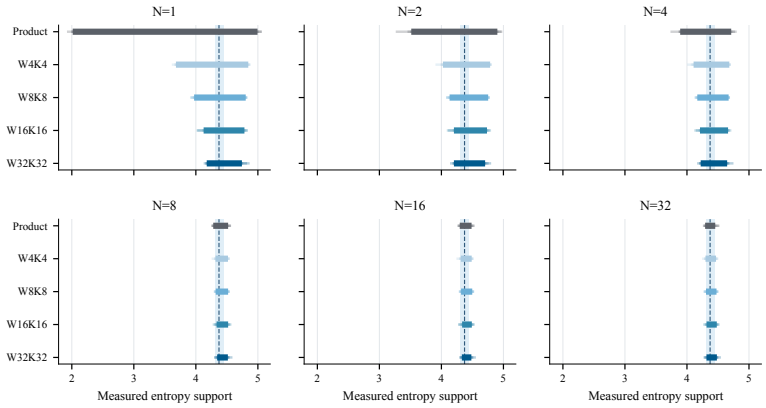



Figure 11: Measured entropy support for every training seed and the resulting three-seed intersections. The dashed line marks the validation mean. The comparison protocol never extrapolates beyond these intervals.

**Path and center-out topologies.** The adjacent path and center-out tree orient the same undirected chain in different ways. The path conditions each position on its immediate clean parent, giving maximal depth $D = W$. Center-out forks around a central separator and advances the two branches in parallel. At the validation mean entropy, the path has 0.3–6.8% lower mean GenPPL across the six reverse budgets.

Center-out remains attractive when latency matters. At $W = 8$ it retains all adjacent edges with $D = 5$ instead of $D = 8$. With unrestricted conditional factors, these two orientations represent the same undirected tree family; center-out does not lose an additional edge or dependence relative to the order-2 path. Their empirical difference concerns orientation, parameterization, and optimization, rather than a strict inclusion of graphical families. The argument does not extend to the complete-joint family. When $K = W$, preserving all previous clean variables still requires $D = W$; forking the state would introduce conditional independences and define a different joint.

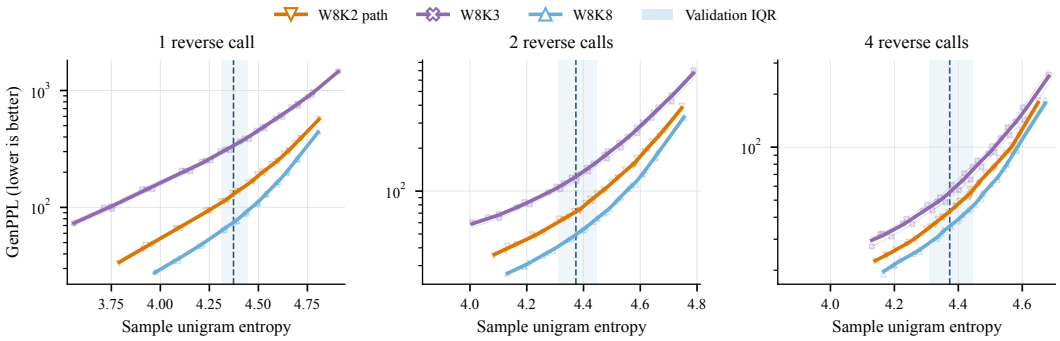



Figure 12: Order at fixed width $W = 8$: $K = 2, 3, 8$, with respective depths $D = 8, 3, 8$, for $N = 1, 2, 4$. The complete local joint is strongest; finite neural realizations need not improve monotonically at every intermediate $K$.

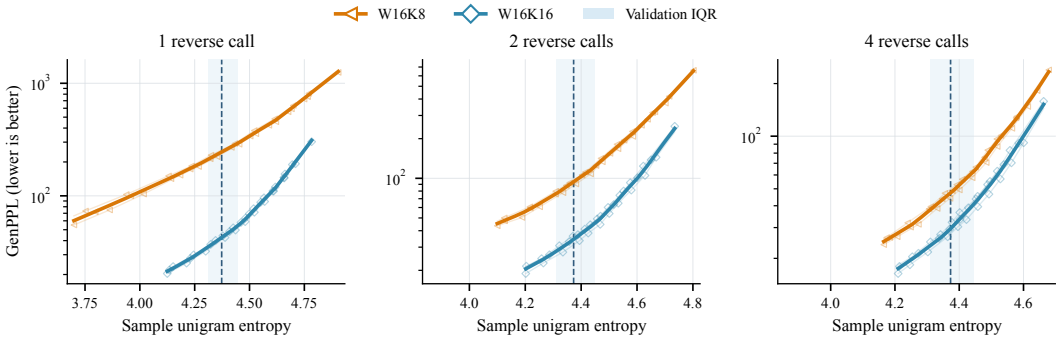



Figure 13: Coverage at fixed width $W = 16$. W16K8 uses depth eight and a shared seven-token separator; W16K16 uses depth sixteen and full-prefix conditioning. This comparison changes both clean-context coverage and depth.

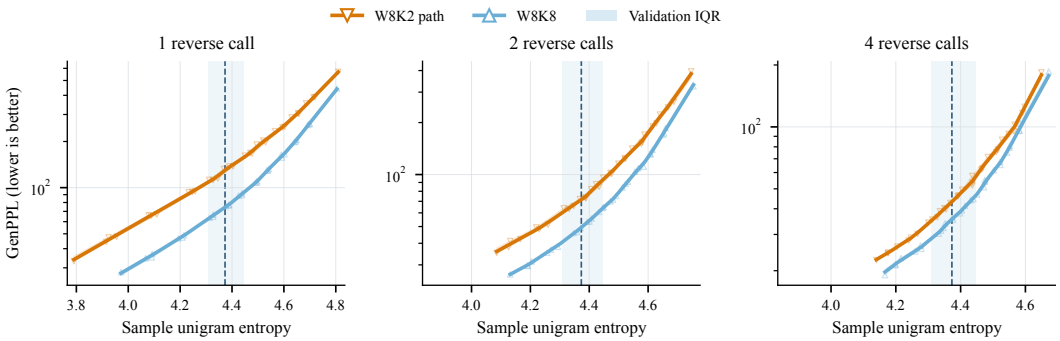



Figure 14: Equal-depth control. W8K2 and W8K8 both use $D = 8$ lightweight rounds, isolating clean-context expressivity from local serial depth.

### C.7 Native Structure versus Step Distillation

Figure 16 compares native Product and W32K32 with a product D-MMD student (Hoogeboom et al., 2026). This control uses one training seed, shared by the displayed native checkpoints. Separate students target $N = 1, 2, 4$, starting from the final Product checkpoint. The frozen Product teacher supplies denoising targets at sampled times. Hard clean samples are stop-gradient, and the categorical prior is transported through the exact finite uniform bridge before applying the distillation loss. Each run uses 10,000 alternating student/auxiliary updates (5,000 each), batch size 64

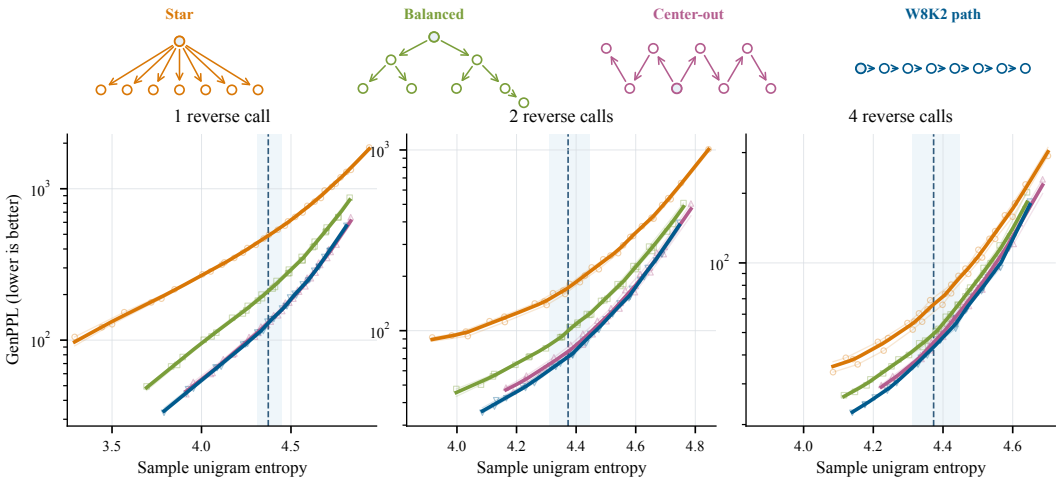



Figure 15: Order-2 topology at fixed $W = 8$. The top row makes the star, balanced, center-out, and adjacent-path factorizations explicit; the bottom row compares their temperature frontiers. Topology matters inside an order class, but none reproduces the complete-joint result.

across four H100s, and AdamW with learning rate $2 \times 10^{-5}$, $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.01, 100-update warmup, and gradient clipping at 1. We evaluate the final student.

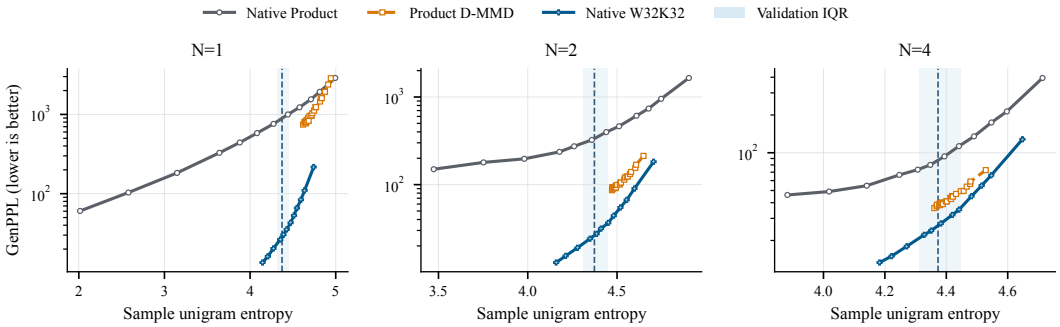



Figure 16: Native Product, product D-MMD, and native W32K32 frontiers at $N = 1, 2, 4$. The validation mean entropy is jointly supported only at $N = 4$; the other panels show measured support without extrapolation.

Interpolating the plotted 256-sample frontiers at $N = 4$ and the validation mean entropy gives GenPPL 87.17, 38.86, and 26.27 for Product, D-MMD, and W32K32. D-MMD recovers much of the few-step Product gap, but does not reach the native structured frontier on their common support.

### C.8 Same-Checkpoint TinyStories Attribution

We evaluate three paths at the validation mean entropy. P/P is the Product checkpoint with its native transition. S/P decodes the W32K32 checkpoint through the Product path, with the structured head disabled. S/S restores the trained structured transition (Table 3).

S/P is worse than P/P at every budget, whereas S/S is better by a wide margin. The W32K32 checkpoint is specialized to its structured transition; its gain is not present in the head-disabled Product path.

| Mode | $N = 1$ | $N = 2$ | $N = 4$ |
|---|---|---|---|
| P/P | $911.14 \pm 16.49$ | $351.66 \pm 15.17$ | $90.37 \pm 2.78$ |
| S/P | $2009.15 \pm 1071.15$ | $1074.47 \pm 486.02$ | $505.03 \pm 173.69$ |
| S/S | $\mathbf{28.35 \pm 0.26}$ | $\mathbf{26.03 \pm 0.64}$ | $\mathbf{26.14 \pm 0.81}$ |

Table 3: TinyStories same-checkpoint attribution at the validation mean entropy. Entries are GenPPL, mean $\pm$ sample s.d. over three training seeds, interpolated from 256-sample temperature sweeps.

## D GIDD-3B Transfer Protocol

**Adaptation.** We initialize from the released `dvruette/gidd-unif-3b` checkpoint (von Rütte et al., 2026) and freeze all pretrained parameters, including the native vocabulary projection. Rank-16 LoRA adapters with scale 32 and no dropout are added to the query, key, value, and output projections of every attention layer and to both MLP projections. SPDLM uses W16K2 adjacent paths ($D = 16$), implemented by the one-parent cache with rank 256 and eight heads; the correction output is zero-initialized. The singleton cache is written from the raw parent backbone state. Propagating its conditioned state would instead carry ancestors and would not implement this K2 factorization. Unlike the TinyStories read in Eq. (17), this adapter uses $u_v = W_o(q_v + W_a a_v)$, where $q_v = W_q \,\mathrm{LN}_{\mathrm{child}}(h_v)$ and $a_v$ is the attended cache value. It adds $u_v$ to the native output state before vocabulary projection.

We train for 10,000 updates on the official 54.1M-token Nemotron pretraining shard released with GIDD. Sequences are packed to length 512 with batch size 8 and no gradient accumulation, so the adapted checkpoint observes 40.96M packed tokens. Each run uses one H100, with the frozen backbone in bfloat16 and trainable LoRA and head parameters in float32. This experiment uses one training seed and the native uniform-corruption protocol: prompt conditioning is selected with probability 0.2 and leaves the prompt prefix clean; with probability 0.2, token positions receive independent diffusion clocks, and otherwise a sequence shares one clock. The objective is GIDD's native clean-prediction surrogate. To retain the pretrained product interface while learning the structured transition, the SPDLM checkpoint minimizes $\mathcal{J}_{\mathrm{mix}} = \frac{1}{2}(\mathcal{J}_1 + \mathcal{J}_K)$ over active suffix positions, where $\mathcal{J}$ denotes the native GIDD surrogate evaluated on the corresponding logits, rather than categorical CE alone. AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay $10^{-4}$, gradient clipping at 1, and a 100-update linear warmup followed by constant learning rates of $5 \times 10^{-5}$ for LoRA and $10^{-4}$ for the structured head.

**Reverse transition and distributional evaluation.** GIDD directly predicts a clean-token law. For a generated clean parent set $S_v$, SPDLM first forms $p_{\theta,v}(x_0^v \mid \mathbf{x}_{0,S_v}, C)$ and applies the same observation correction as the product model,

$$r_v(x_0^v \mid x_t^v, \mathbf{x}_{0,S_v}, C) \propto p_{\theta,v}(x_0^v \mid \mathbf{x}_{0,S_v}, C)\, q_t(x_t^v \mid x_0^v), \tag{22}$$

before passing the complete clean proposal through GIDD's unchanged exact uniform bridge $q(\mathbf{x}_s \mid \mathbf{x}_t, \widehat{\mathbf{x}}_0)$. Setting the structured correction to zero exactly recovers the native product transition.

Head OFF and ON in Section 5.2 use the same adapted checkpoint. We draw 120 distinct held-out Nemotron examples, condition on the first 32 GIDD tokens, and generate 128-token continuations with $N = 8$, nucleus threshold 0.95, temperatures $\{0.7, 0.8, 0.9, 1.0, 1.1, 1.2\}$, and paired prompts and random seeds. We use the centered Gradient Moment estimator of Hoogeboom et al. (2026). Let $g(B)$ be the gradient of the mean continuation-token NLL in a batch $B$ with respect to all GPT-2-large parameters. Each paired batch of eight examples is split into two halves, giving generated batches $B_1^g, B_2^g$ and held-out batches $B_1^r, B_2^r$. Its estimate is $(g(B_1^g) - g(B_1^r))^\top (g(B_2^g) - g(B_2^r))$. Prompt and padding tokens are excluded from the loss; gradients are neither unit-normalized nor divided by the parameter count. GPT-2-large runs in evaluation mode with bfloat16 weights and its own tokenizer, with sequences truncated to 256 tokens. We report the mean of the 15 estimates and their sample standard deviation divided by $\sqrt{15}$ as the standard error. At temperature 1.0, Product and SPDLM have essentially identical sample entropies (4.06 for both), while centered Gradient Moment falls from 4.90 to 3.22. The main figure instead highlights the held-out entropy, $H = 4.16$:

linear interpolation of each measured entropy–Gradient Moment curve gives 6.11 for head OFF and 4.60 for head ON, a 24.8% reduction.

**Latency.** We benchmark the same checkpoint on one H100 at $N = 8$, temperature 0.8, nucleus threshold 0.95, a 32-token prompt, and a 128-token continuation. Model loading is excluded. Each entry is the median synchronized batch time after two warmups and five measured repetitions. Both paths use the compact uniform-bridge implementation; Product disables the structured head.

| Batch | Product time (s/batch) | SPDLM overhead |
|---|---|---|
| 1 | 0.95 | 12.65% |
| 2 | 1.82 | 7.14% |
| 4 | 3.52 | 3.83% |
| 8 | 6.94 | **2.33%** |

# E SUMI-7B: FULL GSM8K STUDY

## E.1 MATCHED TRAINING AND EVALUATION

Product-LoRA and SPDLM start from released Sumi-7B and train for one shuffled epoch over the 7,473 GSM8K examples, with batch size one and no gradient accumulation. Product-LoRA and K2-pure SPDLM use three paired training seeds. Four demonstrations are drawn from the training split. Targets contain the first 128 tokens of the natural rationale and answer after calculator annotations are removed. Paired runs use identical examples, prompts, and corruptions. Each model is evaluated after 7,473 updates.

LoRA (Hu et al., 2021) adapts the attention query/key/value/output and MLP gate/up/down projections in all 36 layers, with rank 16, scale 32, and zero dropout. The remaining pretrained weights, including the vocabulary projection, stay fixed. The merged-attention head has rank 256 and eight heads. AdamW uses learning rates $5 \times 10^{-5}$ for LoRA and $10^{-4}$ for the head, weight decay $0.01$, 100 warmup updates, and unit gradient clipping. Product-LoRA and SPDLM retain Sumi's native uniform corruption, importance weighting, valid-token reduction, and auxiliary z-loss. Pure training evaluates the native surrogate only on structured logits.

SPDLM uses W16K2 raw-parent paths and the query-residual read described in Section D. Sumi's backbone states are already RMS-normalized. With native norm weight $g$, write $a = h/g$ and $r(a) = \sqrt{\mathrm{mean}(a^2) + \epsilon}$. The adapter returns $\widetilde{h} = g \odot (a + u)r(a)/r(a + u)$ before vocabulary projection, preserving the native RMS scale rather than adding an unconstrained residual after normalization. The implementation safeguards division by $g$.

Evaluation covers all 1,319 GSM8K test questions with four-shot prompts, a 2,048-token canvas, 128 generated answer positions, temperature zero, and $N = 1, 2, 4, 8, 16, 32, 64$ ancestral reverse calls. All arms use a shared evaluation seed and identical sharding. Temperature zero makes clean-proposal selection greedy; the uniform reverse bridge remains unchanged. Uniform-CoDD's circuit-specific decoding is described below. Strict exact match requires the #### marker used by the Sumi evaluation protocol. We report three training seeds.

## E.2 THREE-SEED REPLICATION

Table 4 gives the individual runs and the matched same-checkpoint head-OFF control.

Every seed favors SPDLM over Product-LoRA at $N = 8, 16, 32, 64$. The mean advantage decreases from $9.30$ points at $N = 8$ to $4.30$ points at $N = 64$; the corresponding paired-question 95% intervals are $[7.99, 10.64]$ and $[2.70, 5.89]$. At $N = 32$, the three gains are $8.19$, $4.17$, and $5.08$ points.

Intervals use 20,000 paired question-cluster bootstrap replicates, with all methods and training seeds retained for each sampled question. They measure question uncertainty conditional on the trained models; tables report the separate variation across training seeds.

| $N$ | Seed | Product-LoRA | SPDLM OFF | SPDLM ON |
|---|---|---|---|---|
| 8 | 17 | 89 (6.75) | 51 (3.87) | 237 (17.97) |
| 8 | 29 | 74 (5.61) | 44 (3.34) | 163 (12.36) |
| 8 | 41 | 54 (4.09) | 37 (2.81) | 185 (14.03) |
| 16 | 17 | 216 (16.38) | 172 (13.04) | 318 (24.11) |
| 16 | 29 | 155 (11.75) | 102 (7.73) | 229 (17.36) |
| 16 | 41 | 151 (11.45) | 120 (9.10) | 236 (17.89) |
| 32 | 17 | 280 (21.23) | 267 (20.24) | 388 (29.42) |
| 32 | 29 | 235 (17.82) | 187 (14.18) | 290 (21.99) |
| 32 | 41 | 234 (17.74) | 195 (14.78) | 301 (22.82) |
| 64 | 17 | 287 (21.76) | 260 (19.71) | 373 (28.28) |
| 64 | 29 | 250 (18.95) | 204 (15.47) | 307 (23.28) |
| 64 | 41 | 283 (21.46) | 218 (16.53) | 310 (23.50) |

Table 4: Individual GSM8K results: correct answers out of 1,319, with strict accuracy (%) in parentheses. OFF and ON use the same SPDLM checkpoint; OFF disables the added head.

### E.3 Uniform-CoDD Comparator

We port the CoDD probabilistic circuit (Li et al., 2026) to Sumi's uniform diffusion process, using a homogeneous HMM with 1,024 latent states over width-32 components and shared transition and emission probabilities. For each seed, the corresponding Product-LoRA checkpoint is frozen and only the circuit is trained for one additional GSM8K epoch. Uniform-CoDD therefore receives a second epoch of task-specific training; Product-LoRA and SPDLM each receive one.

Let $\pi_v$ be the frozen model's clean-token prior and set $b_v(a) = \pi_v(a) q_t(x_t^v \mid a)$. The port fits the normalized conditional

$$Q_\omega(\mathbf{y}_T \mid \mathbf{x}_t) = \frac{p_\omega^{\mathrm{HMM}}(\mathbf{y}_T) \prod_{v \in T} b_v(y_v)}{Z_\omega(\mathbf{x}_t)}, \tag{23}$$

using loss $-|T|^{-1} \log Q_\omega(\mathbf{x}_{0,T} \mid \mathbf{x}_t)$. The normalizer is computed by scaled HMM forward messages over the full vocabulary. Every noisy coordinate contributes soft channel evidence. Our port optimizes this conditional likelihood with Adam; Li et al. (2026) train their masked-diffusion circuit with EMOne.

Training uses 7,473 updates, batch size one without accumulation, Adam with learning rate $10^{-3}$, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, no weight decay, 100 linear warmup updates, and gradient clipping at 1. The circuit is float32 and the frozen backbone is bfloat16. Initial emission and transition logits are independent zero-mean Gaussians with standard deviation 0.1; initial-state logits are zero. We retain the same training examples, four-shot prompt construction, answer window, and uniform corruption as the matched adapters.

At inference, coupling activates only when $1 - \alpha_t < 0.5$; earlier steps use the product path. Temperature 0.2 scales the neural-prior logits before the untempered channel likelihood is applied. Forward-filtering/backward sampling draws an HMM state path, followed by greedy token selection conditional on that path. The uniform reverse bridge is unchanged. This activation rule leaves the circuit inactive at $N = 1$. The reported accuracy and memory comparisons concern this uniform-noise implementation of CoDD.

At $N = 1$, Uniform-CoDD is identical to Product-LoRA for every seed. Its mean accuracy exceeds Product-LoRA's at $N = 4, 16, 32, 64$, but remains below SPDLM throughout the sweep (Table 5). The SPDLM–CoDD margin is largest at $N = 8$ ($9.75 \pm 1.98$ points) and narrows to $2.83 \pm 0.93$ points at $N = 64$. All three seeds favor SPDLM at each of the seven budgets; aggregate values appear in Table 1.

### E.4 Controlled Throughput and Memory

We time isolated H100 generation using a fixed ordered set of 32 GSM8K prompts; each batch size uses the corresponding prefix of this set, shared across methods. Each cell has 50 synchronized measurements, collected in one or more runs with 10 warmups per run. Precision, canvas, sampler,

| $N$ | Seed | Product-LoRA | Uniform-CoDD | SPDLM | SPDLM–CoDD |
|---|---|---|---|---|---|
| 1 | 17 | 0.23 | 0.23 | 1.97 | +1.74 |
| | 29 | 0.15 | 0.15 | 1.82 | +1.67 |
| | 41 | 0.15 | 0.15 | 2.96 | +2.81 |
| 2 | 17 | 0.53 | 0.30 | 4.70 | +4.40 |
| | 29 | 0.15 | 0.15 | 1.74 | +1.59 |
| | 41 | 0.15 | 0.15 | 2.50 | +2.35 |
| 4 | 17 | 1.44 | 1.82 | 10.99 | +9.17 |
| | 29 | 0.76 | 1.21 | 6.22 | +5.00 |
| | 41 | 0.99 | 1.21 | 8.72 | +7.51 |
| 8 | 17 | 6.75 | 6.37 | 17.97 | +11.60 |
| | 29 | 5.61 | 4.70 | 12.36 | +7.66 |
| | 41 | 4.09 | 4.02 | 14.03 | +10.01 |
| 16 | 17 | 16.38 | 17.29 | 24.11 | +6.82 |
| | 29 | 11.75 | 13.19 | 17.36 | +4.17 |
| | 41 | 11.45 | 12.13 | 17.89 | +5.76 |
| 32 | 17 | 21.23 | 21.99 | 29.42 | +7.43 |
| | 29 | 17.82 | 18.35 | 21.99 | +3.64 |
| | 41 | 17.74 | 18.42 | 22.82 | +4.40 |
| 64 | 17 | 21.76 | 24.64 | 28.28 | +3.64 |
| | 29 | 18.95 | 20.24 | 23.28 | +3.03 |
| | 41 | 21.46 | 21.68 | 23.50 | +1.82 |

Table 5: Seed-level strict GSM8K accuracy (%) for the matched Product-LoRA, Uniform-CoDD, and SPDLM comparison.

tokenizer, and compilation state are fixed; model loading and scoring are excluded. Times are median batch duration divided by batch size and therefore measure amortized throughput, not batch-one response latency.

| Batch | Product-32 | SPDLM-32 | Head overhead | Product/SPDLM mem. | CoDD mem. |
|---|---|---|---|---|---|
| 1 | 7.68 | 8.11 | 5.72% | 16.53 | 17.42 |
| 2 | 7.45 | 7.66 | 2.88% | 18.38 | 19.38 |
| 4 | 7.32 | 7.45 | 1.70% | 22.09 | 23.31 |
| 8 | 7.29 | 7.38 | 1.23% | 29.50 | 31.17 |
| 16 | 7.15 | 7.20 | 0.60% | 44.32 | 46.88 |
| 32 | 7.17 | 7.20 | 0.52% | 73.95 | OOM |

Table 6: Controlled Sumi-7B latency in seconds per question and peak allocated memory in GiB. Product and SPDLM instantiate the same LoRA checkpoint family; Product bypasses the structured head.

At batch eight, SPDLM-32, Product-LoRA-64, and Uniform-CoDD-64 take 7.38, 14.60, and 14.58 seconds per question, respectively.

Product and SPDLM have the same measured peak (Table 6): the full-canvas backbone sets the allocation high-water mark, and the 4.28M-parameter one-slot head remains below it. Our Uniform-CoDD implementation retains a float32 emission table in $\mathbb{R}^{1024\times 100278}$ and normalized unary potentials over $B \times 128 \times 100278$ entries. These tensors raise peak allocation by 5.4–5.8% through batch 16 and exceed the 80-GiB H100 capacity at batch 32.

### E.5 Pure versus Mixed Structured Training

Let $\mathcal{J}_S$ and $\mathcal{J}_P$ denote Sumi's native structured and product diffusion objectives. SPDLM minimizes $\mathcal{J}_{\text{pure}} = \mathcal{J}_S$. The mixed control minimizes $\mathcal{J}_{\text{mixed}} = \frac{1}{2}\mathcal{J}_S + \frac{1}{2}\mathcal{J}_P$ and can therefore also run as a product model with the head disabled. Table 7 compares the two objectives at $N = 32$ using one paired training seed.

Pure training gains 9.17 points when the head is enabled; mixed training gains only 0.83. The mixed objective instead improves the head-disabled Product path from 21.23% to 24.94%. In this comparison, mixed training improves head-OFF accuracy at the expense of head-ON accuracy. Because it also halves the weight on $\mathcal{J}_S$, the table does not isolate the underlying gradient mechanism.

| Objective | Head OFF | Head ON |
|---|---|---|
| K2 pure | 20.24 | **29.42** |
| K2 mixed | **24.94** | 25.78 |
| Product-LoRA | 21.23 | — |

Table 7: Strict exact match at $N = 32$. Pure training gives higher accuracy with the head enabled; mixed training gives higher accuracy with it disabled.

### E.6 Paired Qualitative Example

Table 8 shows the four methods on the same GSM8K prompt at $N = 32$. Generations are reproduced verbatim apart from line wrapping.

**GSM8K prompt**

Mark has $50 in his bank account. He earns $10 per day at his work. If he wants to buy a bike that costs $300, how many days does Mark have to save his money?

**Reference computation:** $(300 - 50)/10 = 25$ days.

**Released Sumi** — Incorrect

```
0 0 0 0 0 000 0 0 0 0 0 0 0 0000000000 0 0 0 0 000 0 0 0 0 0 0 000 0
000 0 0
```

**Product-LoRA** — Incorrect

```
Mark needs to save $300 - $50 = $<<300-50=250>>250 dollars.  ####
250
```

**Uniform-CoDD** — Incorrect

```
Mark needs to add $250 in savings because $300 - $50 = $250.  Mark
needs to save $10 per day for 10 days because $10/$10 = 10 days.
Mark needs to save 10 days + 10 days = 20 days.  #### 20
```

**SPDLM (ours)** — Correct

```
Mark needs to save $300 - $50 = $250 more money because 300 - 50  =
250.  So, Mark needs to save $250 / $10 = 25 days because 250/10 = 25.
#### 25
```

Table 8: Cherry-picked paired qualitative comparison at $N = 32$. Product-LoRA recovers the remaining dollar amount but reports it as the answer; Uniform-CoDD attempts a second arithmetic stage but computes it incorrectly; SPDLM completes both stages and returns the exact answer.