ON-POLICY RESIDUAL LEARNING FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Residual context is emerging as a key ingredient for unlocking the speed–quality Pareto frontier of diffusion language models (DLMs) by carrying forward predictive distributions across denoising steps. Yet turning this seemingly simple mechanism of computation recirculation into an effective and efficient learning paradigm is far from simple: there is no clear recipe for how residual context should be learned. We introduce ORL, a framework for on-policy residual learning in DLMs. Algorithmically, we identify that residual learning is inherently on-policy. ORL learns directly from these model-induced residual trajectories on the fly and introduces an entropy-contraction objective that encourages sharpening predictions as the model revisits its own intermediate predictions. System-wise, exact residual supervision requires capturing, retaining, and repeatedly processing many trajectory-dependent intermediate states. ORL co-designs the learning algorithm with asynchronous residual capture and transport, specialized block-diffusion attention, and packed compute-balanced training, making on-policy residual learning practical at scale. On a 4B DLM converted from Qwen3-4B, ORL improves tokens per forward (TPF) by up to 6.2× and end-to-end throughput by approximately 5× at comparable or better generation quality. On a single H100, ORL-trained models achieve a stronger speed–quality Pareto frontier than autoregressive models accelerated with EAGLE3 and DFlashV1 speculative decoding, under both strict and relaxed verification. On LLaDA-2.1-mini, 36 H100-hours and 4.2M self-generated tokens, about 10× less post-training data than prior LLaDA parallel-decoding recipes, yield up to 1.5× higher TPF and up to 3× higher throughput at matched quality.

1 INTRODUCTION

Today's LLMs collapse the complexity of language generation into an extremely simple next-token prediction (NTP) paradigm. This simplicity delivers remarkable intelligence, but at a fundamental systems cost: autoregressive (AR) decoding remains sequential, severely underutilizing the massive parallelism of modern hardware (Nebius team, 2025; Frye & Biswas, 2026; Robbins, 2026; Killinger & Kiely, 2025). Unlocking parallel decoding for LLMs has emerged as one of the most timely problems in efficient language generation (Almeida, 2026; Liu et al., 2024; Yu et al., 2026; Bie et al., 2026). Among the most actively explored paradigms are discrete diffusion language models (DLMs). By initializing blocks of masked tokens and progressively unmasking them, DLMs transform each forward pass from a single-token prediction into an opportunity to decode many tokens (Nie et al., 2025; Arriola et al., 2025; Ye et al., 2025; Cheng et al., 2025).

Recently, there has also been an interesting debate: given speculative decoding, a lossless acceleration technique, why do we still need diffusion language models? We point out that DLMs open a different optimization dimension for language generation: parallelism that the model itself learns. By contrast, existing LLMs via next-token prediction optimize next-token accuracy. Speculative decoding (Chen et al., 2026a; Cheng et al., 2026; Huang et al., 2026) poses another dimension: it optimizes speed for a fixed target, and the attainable speedup is capped by how predictable the target already is. DLMs instead expose quality and parallelism as a joint learning problem: the model must learn not only what to predict but also how much reliable progress to make in each forward pass. Optimizing both dimensions in a single model is what makes a better speed–quality tradeoff possible than acceleration around a fixed target. It is one unique advantage of a DLM target (Figure 1b).

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.

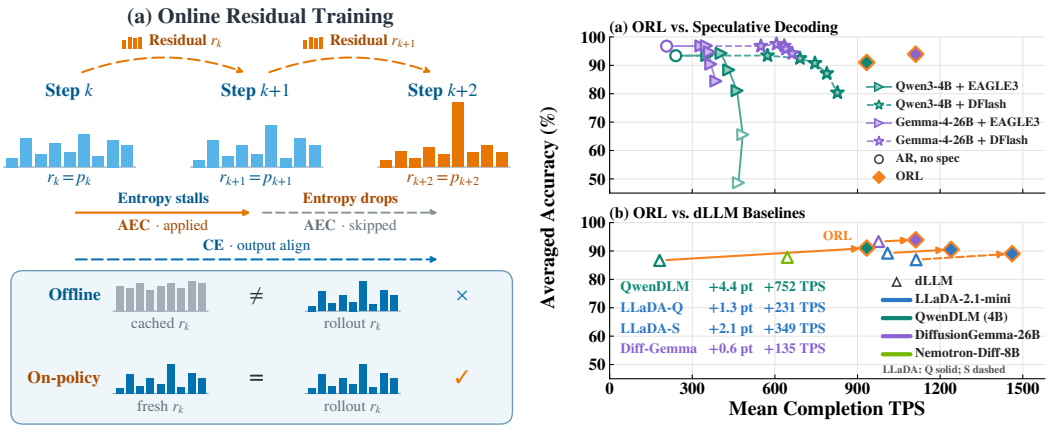


Figure 1: **(a)** On-policy residual training. Residual predictions r_k are carried across denoising steps. ORL trains on the fresh residuals produced by the current rollout (on-policy) rather than on cached residuals that no longer match the rollout (offline), and enforces entropy contraction across steps, applied only where entropy stalls, so that more tokens are committed. **(b)** A DLM target that learns to refine its residuals reaches the speculative-decoding frontier. Average GSM8K/HumanEval quality.

Yet optimizing the speed–quality tradeoff is difficult: beyond deciding which token should fill each position, the model must also determine how aggressively it should fill the block. Existing work tackles this challenge through improved diffusion objectives, decoding strategies, reinforcement learning, and iterative self-refinement (Hu et al., 2026b; DiffusionGemma Team et al., 2026; Lee et al., 2026; Wang et al., 2026b; Yu et al., 2026; Hu et al., 2026a; Ren et al., 2026). Among these directions, one idea is emerging as particularly natural and effective: recirculating residual predictions across decoding steps. In particular, each forward pass already spends substantial computation constructing a predictive distribution over the unresolved block. Rather than discarding this distribution and recomputing from scratch at the next step, residual decoding feeds it back as context, allowing subsequent steps to build upon the model's previous predictions (Hu et al., 2026b; Jo et al., 2026; Lee et al., 2026). Appendix B.1 shows that these intermediate predictions often anticipate the final tokens before they are committed. RCD (Hu et al., 2026b) and Looped Diffusion (Lee et al., 2026) show that this residual information can significantly improve the speed–quality Pareto frontier. DiffusionGemma (DiffusionGemma Team et al., 2026) pushes this trend even further, achieving over 20 tokens per forward (TPF; about 980 tokens/s on one H100, as shown in Figure 1b), significantly faster than existing speculative decoding.

Existing methods, however, largely treat residual information as an architectural or decoding primitive, leaving no clear recipe for learning it effectively and efficiently. Algorithmically, residual learning is hard for two reasons. First, the learning loop is self-referential: Offline training cannot faithfully reproduce the states seen at inference time, namely partially denoised blocks together with the residuals they receive. Even for on-policy training, under a shared optimizer, the residual pathway moves much faster than the backbone, and naïve updates destabilize training. Second, supervising intermediate states toward the final output indicates what each state should predict, but not how much progress each step should make. In practice, we find that entropy stays nearly flat across denoising steps, and TPF barely improves (Figure 2b). On the systems side, exact residual supervision requires generating intermediate predictions in flight and repeatedly feeding these states back to training. This trajectory-dependent workflow fits poorly into existing RL pipelines (§2).

We introduce ORL, an on-policy residual learning framework for DLMs that addresses each of these challenges. **(1)** Residual learning is inherently on-policy, and the loop it closes must be kept stable. ORL learns from residual trajectories generated online by the current model, using the exact residual inputs encountered during rollout (Figure 1a, bottom). It rescales the residual pathway so that it updates at the backbone's pace and gates supervision by rollout entropy, ensuring that already-confident predictions are not sharpened further. **(2)** Refinement speed needs its own objective. ORL introduces an adjacent entropy-contraction objective that explicitly requires each step to reduce the uncertainty left by the previous one, sharpening predictions along the residual trajectory (Figure 1a, top). On the systems side, **(3)** ORL makes exact on-policy residual learning practical at scale. It asynchronously captures residual states off the decoding critical path, uses memory-

adaptive transport to keep or offload them depending on placement, and packs trajectory-dependent states into dense training batches. A compute-aware scheduler balances the resulting irregular workloads across GPUs, while a specialized staircase attention kernel executes packed block-diffusion states without materializing dense masks. Together, these designs make residual training efficient.

Contributions. To the best of our knowledge, ORL represents the first open, end-to-end workflow for exact on-policy residual optimization in DLMs (see the Reproducibility statement):

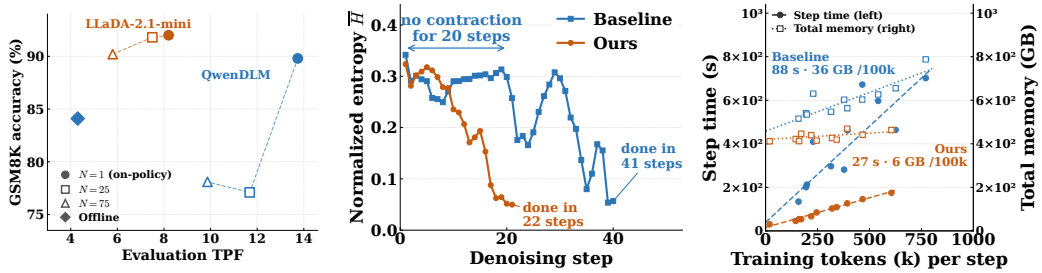
- **We identify residual refinement as an inherently on-policy learning problem.** Existing residual training suffers from two mismatches: offline residuals can induce trajectories different from those encountered at inference, while pre-generated trajectories become stale as the model evolves. This reveals that residuals must be optimized on the evolving state distribution they induce, with the residual pathway updated slowly enough that the loop stays stable.
- **We introduce a general recipe for on-policy residual optimization.** ORL trains intermediate states from fresh rollouts using their exact residual conditioning, distilling them toward verified final outputs, and explicitly improves step-wise refinement through adjacent entropy contraction with stabilized residual updates. The same recipe supports direct post-training of existing DLMs, without requiring a model-specific training pipeline.
- **We make exact trajectory-level residual optimization practical at scale.** We co-design the rollout and training system, including asynchronous in-engine trajectory capture, memory-adaptive residual transport, specialized block-diffusion attention, packed intermediate-state training, and compute-aware GPU balancing. These techniques speed up training by up to $5.2\times$.
- **We establish a stronger speed–quality frontier with little post-training.** With 60 H100-hours of post-training on QwenDLM, a 4B DLM converted from Qwen3-4B, ORL improves TPF by 2.5–$6.2\times$ and throughput by approximately $5\times$ at comparable quality; in single-request (batch-size-1) serving on one H100, the regime speculative decoding targets, the resulting model is faster than every EAGLE3 (Li et al., 2025) and DFlash setting we measure: $1.25\times$ faster than DFlash at matched accuracy with relaxed verification, and $1.63\times$ faster than DFlash with strict verification, which keeps the AR's accuracy, 2.4 points above ours (Figure 1b). The same recipe improves LLaDA-2.1-mini by up to $1.5\times$ higher TPF and up to $3\times$ higher throughput, and DiffusionGemma by 1.1–$1.2\times$ higher TPF. ORL achieves these benefits with a few million self-generated tokens: on LLaDA-2.1-mini it uses 4.2M tokens and 36 H100-hours, about $10\times$ less data than the roughly 40M tokens of d3LLM-LLaDA and dParallel (Figure 5; Appendix H.5).

2 Related Work and Motivation

Residual diffusion decoding. A block DLM generates a response block by block: each block of B tokens is denoised over several steps, conditioned on the prompt and the previously committed blocks, which we call its *clean context*. At step k, the backbone f_θ observes a discrete canvas $s^{(k)} = (x^{(k)}, m^{(k)})$, where $x^{(k)} \in \mathcal{V}^B$ holds the current tokens and $m^{(k)} \in \{0,1\}^B$ marks the unresolved positions, which hold noise tokens (a mask token or random vocabulary tokens), and predicts a distribution $p_i^{(k)}$ over the vocabulary $\mathcal{V}$ for every position. A decoding rule then accepts the unresolved positions whose predictions are confident enough, for example under a probability threshold or an entropy budget, may edit accepted ones, and re-noises the rest; the block is committed once no position remains unresolved or a step cap is reached. The number of steps K of a block therefore depends on the model and the decoding rule, and TPF counts completion tokens per denoising forward. Residual decoding additionally carries the previous step's predictions forward as continuous context (Hu et al., 2026b; Jo et al., 2026; DiffusionGemma Team et al., 2026). Let $r_i^{(k)}$ denote the residual at position i and E_v the input embedding of token v. We inject its expected token embedding through a lightweight residual adapter g_ϕ with injection strength α:

$$\tilde{e}_i^{(k)} = E_{x_i^{(k)}} + \alpha\, g_\phi\big(z_i^{(k)}\big), \qquad z_i^{(k)} = \textstyle\sum_{v\in\mathcal{V}} r_{i,v}^{(k)} E_v, \qquad k > 1. \tag{1}$$

Here $r^{(1)} = 0$, and each position receives the distribution it produced at the previous step. For backbones with next-token (shifted) alignment, such as QwenDLM, the output at position i predicts position $i{+}1$, so $r_i^{(k+1)} = p_{i+1}^{(k)}$; same-position backbones, such as LLaDA, use $r_i^{(k+1)} = p_i^{(k)}$. The adapter is model-specific: QwenDLM uses a gated MLP warmed up offline, LLaDA a zero-initialized gated MLP added for ORL, and DiffusionGemma its native residual module (Ap-



(a) **Residual learning is on-policy.** (b) **Accuracy alone does not accelerate refinement.** (c) **Exact on-policy training is systems-heavy.**

Figure 2: **Three observations behind ORL. (a)** Only fresh trajectories jointly improve accuracy and TPF. **(b)** Standard residual supervision leaves entropy nearly flat across denoising steps. **(c)** Exact on-policy training incurs substantial time and memory overhead.

pendix C). We call $\tau_{\theta,\phi} = \{(s^{(k)}, p^{(k)}, r^{(k)})\}_{k=1}^{K}$ a *residual trajectory*. We call the pair $(s^{(k)}, r^{(k)})$, a canvas together with the residual it receives, the *state* at step k, and index the states of a response by block and step, (b, k). The adapter and the residual it injects form the *residual pathway*.

Residual learning is inherently on-policy. Residual predictions directly affect which tokens are committed, which positions remain unresolved, and therefore which states the model encounters next. As the model changes, the residuals it produces and the trajectories induced by those residuals change together. This makes offline and stale residual supervision fundamentally mismatched. Offline residuals (Jo et al., 2026; Hu et al., 2026b), which are not taken from the current model's own previous denoising step, may induce states different from those encountered by the model at inference time, while pre-generated trajectories (Ren et al., 2026) become stale as optimization updates the model. We define staleness N as optimizer steps between rollout-policy syncs (N=1 is on-policy; offline uses fixed cached trajectories). Existing on-policy DLM methods instead optimize token-space trajectories or final outcomes without directly learning the residual pathway (Wang et al., 2026b; Hu et al., 2026a). Figure 2(a) shows that offline training barely improves TPF, while increasing staleness sacrifices accuracy; only fresh trajectories improve both. Appendix B.2 shows that training keeps changing the model's behavior, with rollout entropy decreasing steadily without collapse, so trajectories from earlier checkpoints become stale.

Accuracy supervision does not imply faster refinement. On-policy trajectories determine *where* residual learning should happen, but not *what behavior* it should encourage. Standard supervision trains each state toward the correct output, yet provides no pressure for one denoising step to make more progress than the previous one. In practice, we observe that the average normalized entropy of the unresolved positions (Section 3.1) remains nearly flat across many steps, even when the final answer is correct (Figure 2b). The model learns to predict accurately, but not to become progressively more decisive. For parallel decoding, this distinction is critical: improving final accuracy does not necessarily increase the number of tokens that can be committed per forward pass. Residual learning, therefore, needs an explicit objective for *refinement speed*, not accuracy alone.

Exact residual learning creates a new systems workload. Correct on-policy supervision requires training on the exact states seen during rollouts. These states are generated dynamically at every denoising step, must cross the rollout–training boundary without perturbing generation, and produce examples with highly irregular trajectory lengths and compute costs. This workload fits poorly into conventional RL training pipelines (Sheng et al., 2024; Fu et al., 2025). A naïve implementation repeatedly materializes these states, wastes memory on sparse trajectory structures, and suffers severe load imbalance. As Figure 2(c) and Appendix B.1 show, its time and memory costs grow rapidly. Thus, **on-policy residual learning requires a system around the states of residual trajectories.**

Appendix A discusses additional related work.

3 Method

Previous research discussed in Section 2 drew three main conclusions: residuals must be learned on the trajectories the current model induces (Figure 2a), refinement speed needs its own objective (Figure 2b), and exact on-policy supervision is a systems workload in its own right (Figure 2c). ORL

answers all three within a loop (Figure 9 in Appendix D). In each iteration, the model generates fresh residual-conditioned trajectories, a verifier keeps the successful rollouts, and ORL replays their states, re-evaluating each with the exact residual input seen during generation. Two objectives are applied: verified-output supervision teaches *what* each state should predict, and adjacent entropy contraction teaches *how quickly* predictions should sharpen across denoising steps. As every update changes the conditioning that generates the next batch, we control how fast the residual pathway moves (Section 3.1). Section 3.2 describes the training system that makes exact replay practical.

3.1 On-Policy Residual Learning

Because residuals and the states they induce change together with the model (Figure 2a), ORL regenerates its training data at every optimization step. At optimization step u, the model (θ_u, ϕ_u) generates G rollouts for each prompt c in minibatch $\mathcal{B}$. Each rollout $\mathcal{R}$ yields a final response $y^{\mathcal{R}}$ and its residual-conditioned trajectory. A verifier $V(c, y) \in \{0, 1\}$ keeps the successful rollouts $\mathcal{R}^{+}_{\mathcal{B}}$; rejected rollouts contribute no gradient. Each nonempty verified batch yields one optimizer step; its trajectories are then discarded, and the updated model generates the next batch. Within each kept rollout we train a subset $\mathcal{S}(\mathcal{R})$ of its states; for QwenDLM, for example, these are the 25% of states in each block that accept the most new tokens, plus the block's final state (Appendix E).

Equation 1 indicates that every step is conditioned on the previous step's prediction, so the canvas alone does not identify a state, and recovering the residual actually consumed would mean re-executing the earlier steps. ORL therefore preserves the *exact states* that the current model visited. For each selected state (b, k), we capture its canvas and the pre-adapter residual $z_i^{(b,k)}$ it consumed, computed with the rollout embedding table $E^{(u)}$. Replay evaluates Equation 1 on the captured canvas with $z_i^{(b,k)}$ held fixed by a stop-gradient, $\mathrm{sg}[z_i^{(b,k)}]$, so gradients reach the backbone and the adapter but not the embeddings through z. Replay places the captured canvas after its clean context and runs one forward pass under block-diffusion attention (Section 3.2), producing the replayed prediction $\widehat{p}_i^{(b,k)}$; model-specific normalization of the injected term, identical in rollout and replay, is omitted from Equation 1 (Appendix C). We store z rather than a vocabulary-sized distribution, and gradients update the backbone and the adapter at the captured state without backpropagating through earlier steps. Exactness here means preserving the captured input, not bitwise equality between rollout and replay outputs. Unlike experience replay in off-policy RL, each captured state is replayed once, by the parameters that generated it, and then discarded. Capturing these inputs without slowing generation is the first requirement of the training system in Section 3.2.

Learning what to predict. For every selected state of a verified rollout, we supervise the replayed prediction $\widehat{p}_i^{(b,k)}$ toward $y^{\mathcal{R}}_{b,i}$, the token at position i of block b in the verified response, with cross-entropy (CE), $\ell^{\mathrm{pred}}_{\mathcal{R},b,k,i} = -\log \widehat{p}^{(b,k)}_{i,\, y^{\mathcal{R}}_{b,i}}$, at both accepted and unresolved positions alike, so that an intermediate prediction corrected later learns directly from its eventual value. Token losses are averaged over the valid positions of each verified response and then across responses, giving $\mathcal{L}_{\mathrm{pred}}$, so that every verified response carries equal weight (Appendix C). Endpoint supervision alone, however, does not indicate how quickly a state should reach its prediction: a model can become more accurate while making little progress from one step to the next. This is the stall in Figure 2(b).

Learning how fast to refine. *Adjacent entropy contraction* (AEC) supplies the missing pressure by asking each step to reduce the uncertainty left by the previous one. Let $\mathcal{U}_{b,k} = \{i : m_i^{(b,k)} = 1\}$ be the unresolved positions, $\overline{H}(p) = -\sum_v p_v \log p_v / \log |\mathcal{V}|$ the normalized token entropy, and $H_{b,k} = \frac{1}{|\mathcal{U}_{b,k}|} \sum_{i \in \mathcal{U}_{b,k}} \overline{H}(p_i^{(b,k)})$ the average unresolved entropy of a state, computed from the detached rollout distribution for the source state ($H^{\mathrm{roll}}_{b,k}$, captured during generation) and from the replayed prediction $\widehat{p}$ for the next state ($H^{\theta,\phi}_{b,k+1}$). For a target contraction rate $\rho \in [0, 1]$,

$$\ell^{\mathrm{AEC}}_{\mathcal{R},b,k} = \Big[\mathrm{ReLU}\Big(H^{\theta,\phi}_{b,k+1} - (1-\rho)\ \mathrm{sg}\left[H^{\mathrm{roll}}_{b,k} \right] \Big) \Big]^2. \tag{2}$$

AEC rests on two design choices. First, it is a hinge: the loss is zero once the next state's unresolved entropy has dropped by at least the fraction ρ, and it penalizes only steps that fail to contract. A uniform entropy penalty would instead keep sharpening every prediction, including those that already make progress, at the risk of trading accuracy for confidence; the hinge acts only where

refinement stalls (Figure 1a). Second, it targets the positions that remain unresolved after acceptance, since each entropy is taken over its own step's unresolved set. The only information the next step receives from the previous one is the tokens just accepted or edited and the residual carrying the previous step's predictive distribution. Thus, AEC rewards the model for turning that residual into confidence on the positions still to be decoded. Pair losses over $\mathcal{P}(\mathcal{R})$, the within-block adjacent pairs (b, k) whose next state $(b, k{+}1)$ is selected and whose two unresolved sets are nonempty, are averaged with equal weight per response into $\mathcal{L}_{\mathrm{AEC}}$ (Appendix C), and the full objective is $\mathcal{L}_{\mathrm{ORL}} = \mathcal{L}_{\mathrm{pred}} + \lambda_{\mathrm{AEC}}\mathcal{L}_{\mathrm{AEC}}$ with $\lambda_{\mathrm{AEC}} \geq 0$. This objective trains *what* each state predicts and *how efficiently* the trajectory reaches it. As shown in Figure 2(b), AEC cuts decoding from 41 to 22 steps with the same answer.

Stabilizing the on-policy feedback loop. Because ORL trains on its own outputs, the loop can fail in two ways, and we guard against each. *The residual pathway can outrun the backbone.* Its outputs are the inputs of the next rollout, yet under a shared optimizer its relative update is about $38\times$ the backbone's, and QwenDLM's verifier reward collapses after roughly 250 training steps (Figure 6; Appendix H.2). We therefore keep the pathway's relative update size close to the backbone's: a pathway warmed up offline is rescaled by γ without changing its function, which under Adam shrinks its relative updates by about γ, and a pathway added from scratch starts from a zero output projection with a lower learning rate (Appendix C). *Supervision can reinforce what the model already believes.* Every CE target is a token the model generated itself, so at positions it already predicts confidently, CE only sharpens an existing belief, and it does so again at every iteration. We therefore apply CE only where the rollout is uncertain, with normalized entropy above a threshold ζ, and keep confident positions near their rollout prediction with a capped reverse KL.

3.2 Efficient Exact Residual Training

This loop is simple to describe but expensive to run. Exact replay turns each response into many states that must be captured during rollout, moved across the rollout–training boundary, and re-evaluated under block-diffusion visibility with highly irregular cost (Figure 2(c)). ORL removes these costs in the order data flows through the training loop (Figure 9), with three mechanisms that change how training executes, but not which states are supervised or which objective is optimized.

Asynchronous exact-state capture. The first cost is getting states out of the decode loop. Once generation advances, recovering an exact residual-conditioned input would require re-executing the preceding steps, while copying each state out synchronously would stall decoding. The engine instead snapshots the canvas, the pre-adapter residual z, and the rollout statistics the objectives need (the source entropies for AEC and, when the entropy gate is used, the output-head inputs from which its reference is rebuilt) at the moment they are produced, enqueues GPU-side handles, and transfers them off the decode critical path. When rollout and training are colocated, the block-local buffers holding z, which we call residual pages, stay GPU-resident and are handed off through CUDA-IPC rather than round-tripping through host memory (Appendix D.1).

Packed and cost-balanced state training. Captured states form the irregular workload of Section 2. Each verified rollout contributes a variable number of states and AEC pairs with different context lengths and residual footprints, so evaluating them one by one recomputes shared clean context and leaves GPUs idle. Following the slicing of TraceRL (Wang et al., 2026b), a row holds a response's clean tokens once together with the states of its blocks at the same step k, so each state reuses the clean context before it. We pack rows from different responses as independent documents, keep residuals as sparse residual pages rather than full-sequence tensors, and assign rows to data-parallel groups in order of largest estimated cost first. Each group returns the numerators and denominators of the response-normalized losses, which are summed globally before division, so packing and redistribution leave the objective unchanged up to floating-point rounding (Appendix D.3).

Staircase attention kernel. Packing leaves one obstacle: each packed row mixes documents, clean context, and noisy blocks, and replay must still respect block-diffusion visibility. A noisy block attends to the prompt, the preceding clean blocks, and itself, but never to its own clean copy, which is present in the row as context for later blocks, to later blocks, or to other noisy blocks; clean queries never attend to noisy keys. A dense mask costs $O(N^2)$ storage and dispatches fully invisible tiles. Our tiled Triton kernel encodes the rule implicitly from per-document metadata, skips invisible

Table 1: **Generation quality and parallel decoding efficiency.** We report benchmark scores (%) and tokens per forward (TPF). Our results evaluate a single checkpoint across all benchmarks. Base and ORL use their selected family-specific decoders (Appendix E.1), so arrows compare complete model–decoder systems rather than weights alone. Within each model-family block, the best result is **bolded** and the starting point for ORL is underlined. Red arrows indicate ORL's TPF improvement.

Model / training	GSM8K		MATH-500		HumanEval		MBPP	
	Acc.↑	TPF↑	Acc.↑	TPF↑	pass@1↑	TPF↑	pass@1↑	TPF↑
Qwen-based models and other DLM baselines								
d3LLM–Dream	81.4	4.9	38.2	3.9	57.1	3.2	55.6	3.0
Fast-dLLM-v2-7B	77.5	2.2	48.7	2.6	61.7	2.6	50.1	2.0
RCD–SDAR-4B-b64	87.0	—	68.4	—	—	—	—	—
TraDo-4B (TraceRL, dynamic)	90.3	—	71.8	—	—	—	—	—
SDAR-8B-b32	88.9	2.9	63.6	4.8	73.5	2.4	58.0	2.4
QwenDLM (4B)	90.4	2.4	88.0	2.1	82.9	1.8	79.2	2.1
+ ORL	**92.4**	**5.9** ↑+3.5	**92.2**	**5.8** ↑+3.7	**89.6**	**11.1** ↑+9.3	**79.4**	**8.0** ↑+5.9
LLaDA-based models								
Fast-dLLM–LLaDA	74.7	2.8	30.8	2.0	37.8	2.6	38.6	2.1
dParallel–LLaDA	72.6	5.1	30.2	3.2	39.0	4.9	40.0	2.4
D2F–LLaDA	73.2	2.9	28.7	2.4	36.6	2.7	38.0	1.9
d3LLM–LLaDA	73.1	**9.1**	30.4	5.7	39.6	6.0	40.6	4.2
dOPSD–LLaDA-8B-Instruct	72.87	—	36.00	—	39.63	—	39.54	—
dUltra-coding	81.5	8.4	35.6	6.7	35.9	7.2	37.0	7.3
DMax-Math	**92.1**	5.5	75.4	5.9	—	—	—	—
LLaDA-2.1-mini	91.9	5.6	84.6	8.2	86.6	11.3	**77.5**	7.7
+ ORL	92.0	8.5 ↑+2.9	**87.0**	**10.2** ↑+2.0	**89.0**	**13.1** ↑+1.8	77.3	**10.3** ↑+2.6
DiffusionGemma-based models								
DiffusionGemma	95.8	26.0	**92.3**	23.7	90.9	24.5	87.7	23.1
+ ORL	**96.0**	**30.0** ↑+4.0	**92.3**	**27.4** ↑+3.7	**91.9**	**27.9** ↑+3.4	**88.2**	**27.0** ↑+3.9

tiles, and supports execution over packed variable-length documents with grouped-query attention (Appendix D.2). Figure 3 measures the cumulative effect of the three mechanisms.

4 Experiments

Evaluation Methodology. We evaluate ORL from two popular starting points on nine benchmarks. For *AR start*, we convert Qwen3-4B-Thinking-2507 (Yang et al., 2025) into a residual-conditioned DLM, termed QwenDLM, through an SDAR-style offline training pipeline, and continue optimization with ORL. For *DLM start*, we post-train LLaDA-2.1-mini and DiffusionGemma (DiffusionGemma Team et al., 2026). Full training and evaluation details are in Appendix E.

4.1 Main Results

ORL substantially improves both quality and parallelism. Table 1 compares ORL with prior DLM baselines, including TraceRL's TraDo-4B (Wang et al., 2026b) and the on-policy self-distillation method dOPSD (Dat et al., 2026).[1] From QwenDLM, it improves TPF by 2.5–6.2× while preserving or improving quality. On HumanEval, for example, ORL reaches 89.6 pass@1 at 11.1 TPF, compared with 73.5/2.4 for SDAR-8B. The same advantage transfers to existing DLMs: on both LLaDA-2.1-mini and DiffusionGemma, ORL raises TPF on all four benchmarks while maintaining or improving quality. For example, on HumanEval, ORL improves LLaDA-2.1-mini from 86.6/11.3 to 89.0/13.1 in pass@1/TPF. On GSM8K and MATH-500, it reaches 92.0/8.5 and 87.0/10.2 in accuracy/TPF, compared with 92.1/5.5 and 75.4/5.9 for DMax-Math (Chen et al., 2026b). Appendix G compares additional methods that are not included in Table 1 because they lack public checkpoints or cannot be evaluated under aligned settings.

[1] Baseline results are quoted from published papers; the MATH-500 entries of rows from the d3LLM evaluation are MATH (4-shot) on the full test set. Appendix E.1 lists the source of each row.

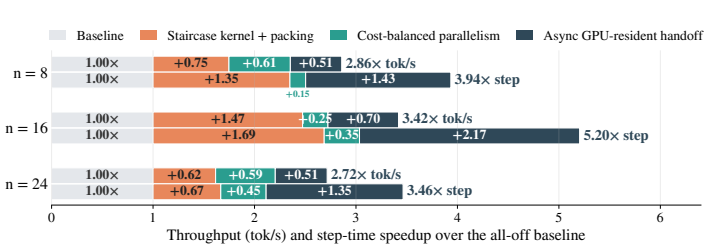


Figure 3: **Cumulative system optimizations.** End-of-bar values report system speedups in training-step time and token throughput.

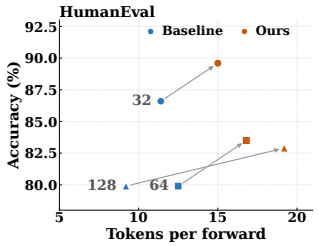


Figure 4: **Scaling with training block size.**

ORL generalizes across models and domains. Benchmarks in Table 1 cover standard math and code evaluations but do not establish whether the same gains transfer broadly. Table 2 evaluates direct post-training of LLaDA-2.1-mini and DiffusionGemma on five additional benchmarks spanning mathematics, knowledge-intensive reasoning, code generation, and instruction following. Across two independently developed DLMs, ORL raises TPF on all ten model–benchmark pairs ($1.26\times$ on average, up to $1.46\times$), with little quality change.

Table 2: **Generalization across domains.** Post-training of LLaDA-2.1-mini and DiffusionGemma (DG) across additional benchmarks. Each entry reports *quality / TPF*.

Dataset	LLaDA Base	LLaDA +ORL	DG Base	DG +ORL
AIME-24	**50.0**/5.31	48.3/**7.77**	55.8/18.15	**56.3/22.02**
MMLU-Pro	57.7/3.60	**58.2/4.88**	**78.6**/18.57	78.1/**22.60**
GPQA-D	**49.5**/3.05	46.1/**4.21**	**65.7**/17.50	65.3/**21.27**
LCB-v6	28.0/5.90	**30.3/7.43**	**53.5**/19.96	51.4/**24.09**
IFEval	**84.1**/1.53	82.3/**1.74**	**66.9**/9.90	**66.9/11.68**

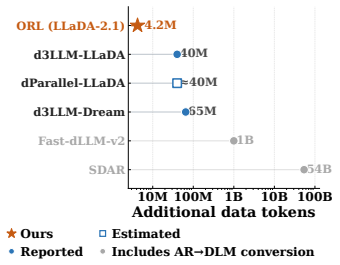
ORL improves the measured batch-size-1 quality–throughput frontier. Because TPF ignores the per-forward cost of residual conditioning and block-diffusion execution, we also measure output throughput on one H100 at batch size 1 (Figure 1(b)); Appendix F details our output-preserving inference optimizations. QwenDLM trained with ORL reaches 933 tok/s at 91.0% average GSM8K/HumanEval accuracy, approximately $5\times$ faster than its starting DLM. AR decoding reaches 240 tok/s. Among measured speculative settings with mean accuracy at least ORL's minus 0.5 percentage points, the fastest EAGLE3 (Li et al., 2025) and DFlash settings reach 405 and 746 tok/s using relaxed verifier thresholds (τ=0.1 and 0.03); with strict verification (τ=1), they reach 353 and 573 tok/s. These are system-level comparisons using each method's native decoder and clean-request throughput; Appendix G reports the full accuracy, failure-count, and verification-threshold context. Sweeping the decoding threshold further shows that ORL shifts the whole quality–throughput frontier outward rather than selecting a more aggressive threshold, improving throughput at selected comparable-accuracy operating points by up to $5.76\times$ on QwenDLM and $2.99\times$ on LLaDA-2.1-mini (Appendix G.4). Appendix J characterizes these accuracy–TPF curves. Unlike speculative decoding, whose speedup is capped by how predictable a frozen target already is, ORL optimizes both speed and quality, with advantage widening with block size (Figure 4).

Rollout freshness changes learning dynamics. Our multi-seed study compares every-step rollout refresh ($N = 1$) with infrequent refresh ($N = 25, 75$), using the same initialization, verifier, and optimization objective (Figure 2a). All settings generate new trajectories; N controls rollout-weight refresh. After converged, the fresh-rollout setting reaches 89.8% mean GSM8K accuracy and 13.7 TPF, versus 77.1%/11.7 and 78.1%/9.9 for $N = 25$ and $N = 75$, respectively. Fresh rollouts start lower but overtake the infrequent-refresh variants. These results motivate frequent refresh for the evaluated recipe. Appendix H.4 reports the per-seed results.

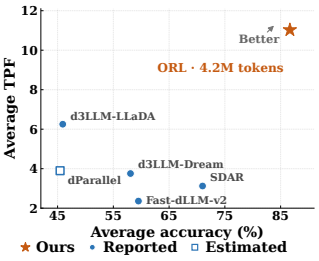
4.2 Ablations and Analysis

System co-design makes exact on-policy training practical. Figure 3 cumulatively enables staircase attention and packed state training, compute-aware balancing, and asynchronous GPU-resident handoff. Across $n = 8/16/24$ rollouts, the complete system reduces optimizer-step time by $3.9/5.2/3.5\times$ and raises token throughput by $2.9/3.4/2.7\times$ over the all-off implementation. Detailed phase breakdowns are provided in Appendix H.1.

ORL is highly data-efficient. Figure 5(a) compares the amount of additional training data used, and Figure 5(b) the quality–TPF operating point it reaches. On LLaDA-2.1-mini, ORL consumes



(a) Data efficiency



(b) Performance comparison

Figure 5: **(a–b)** With only a small amount of self-generated on-policy training data, ORL rapidly improves refinement efficiency and achieves a stronger quality–TPF tradeoff.

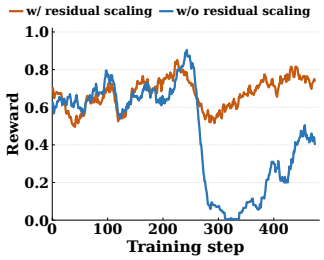


Figure 6: **Residual scaling preserves stable long-horizon optimization.** QwenDLM trained with ORL on GSM8K prompts.

Table 3: **Each component matters, and performance is robust to hyperparameters.** Accuracy (%) / TPF on GSM8K and MATH-500. All variants train QwenDLM with ORL on GSM8K.

(a) Training-design ablation			(b) Hyperparameter sensitivity		
Configuration	GSM8K	MATH-500	Configuration	GSM8K	MATH-500
Full	**94.5 / 6.25**	**90.0 / 5.64**	**Default**	**94.5 / 6.25**	**90.0 / 5.64**
− AEC	92.6 / 4.48	92.6 / 4.38	$\lambda_{\text{AEC}} = 12.5$	94.6 / 5.53	92.8 / 5.06
− entropy-gated RKL	87.8 / 3.29	76.8 / 3.46	$\lambda_{\text{AEC}} = 200$	92.6 / 8.53	88.2 / 7.42
− residual pathway	92.0 / 3.68	75.0 / 3.56	RKL cap $= 0.1$	93.6 / 6.51	92.2 / 5.83
			RKL cap $= 20$	93.8 / 6.80	93.2 / 5.89

4.2M tokens, the prompts and responses of the 5,000 rollouts it generates over 625 training steps (one prompt and eight rollouts per training step), about $10\times$ less than the roughly 40M tokens of dParallel and d3LLM-LLaDA. ORL still reaches a higher average TPF at higher average accuracy than every LLaDA-, Dream-, and Qwen-based baseline. The run takes only 4.5 hours on a single node with eight H100 GPUs (36 H100 GPU-hours), and training ORL on QwenDLM takes 60 H100 GPU-hours. Appendix H.5 details how token counts and GPU-hours are measured.

Residual scaling stabilizes training without sacrificing optimization progress. Over the first 250 training steps, verifier reward is nearly identical with and without scaling. The difference emerges over longer training. With scaling, reward remains high at 0.677, whereas the unscaled run collapses to 0.225 (Figure 6). Removing scaling changes the backbone update magnitude by only $0.96\times$, but increases residual-adapter updates by $31.3\times$ (Appendix H.2).

ORL scales with available refinement capacity. Figure 4 varies the training block size of LLaDA-2.1-mini from 32 to 128. On HumanEval, TPF improves from 11.4 to 15.0 at block size 32 and from 9.2 to 19.2 at block size 128. MBPP shows the same trend (Appendix H.2).

Effective and robust training. Table 3 reports the contribution of each training component and the sensitivity to the main optimization hyperparameters. Removing AEC cuts TPF by 28% on GSM8K and 22% on MATH-500 while changing accuracy by -1.9 and $+2.6$ points, so AEC mainly buys speed, and λ_{AEC} sets this trade-off (Table 3(b)); removing entropy-gated reverse KL or the residual pathway degrades both accuracy and TPF, costing up to 15 points on MATH-500. AEC also gives a better accuracy–TPF tradeoff than a naive entropy loss at different weights (Appendix H.3). In contrast, ORL is robust to its hyperparameters: lowering the AEC weight by $4\times$ or varying the RKL cap over a $200\times$ range lowers accuracy by at most 0.9 points and mainly shifts TPF.

5 Conclusion

We introduced ORL, a framework for on-policy residual learning in DLMs. ORL trains on the verified residual-conditioned trajectories that the current model induces, adds an entropy-contraction objective that sharpens predictions across denoising steps, and co-designs the training system so that exact on-policy supervision is practical. From QwenDLM, our 4B AR-converted DLM, ORL improves TPF by 2.5–$6.2\times$ and throughput by approximately $5\times$ at comparable quality, above the measured EAGLE3 and DFlash operating points. Applied to LLaDA-2.1-mini and DiffusionGemma, it improves TPF on all four main benchmarks, using only 4.2M self-generated tokens.

AI USE STATEMENT

In this work, we used generative AI tools primarily to assist with language polishing and code design. The research ideas, methods, and original manuscript were developed and written by the authors. We reviewed all AI-assisted work and take full responsibility for the final content of this paper, including its text, code, technical claims, and reported results.

ETHICS STATEMENT

This work uses only publicly available, open-source datasets. Our training focuses on improving generation quality and computational efficiency without introducing objectives or supervision intended to shape ethical preferences or value judgments.

REPRODUCIBILITY STATEMENT

We provide an anonymous repository for this work at https://anonymous.4open.science/r/22fb871b-9d65-4fd1-b523-6aa3a47f2857/. Its `configs/paper/experiments.yaml` indexes training recipes, decoder profiles, evaluation matrices, runtime, and scoring records; `docs/reproduction/SETTINGS.md` provides a reading guide. Section 4 and Appendix E summarize the experimental design and aggregation. Appendix D describes exact-state capture, staircase attention, and packed training, while Appendix H.1 reports system measurements. Training data and compute accounting are provided in Appendix H.5.

REFERENCES

AIME. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2024.

Diogo Almeida. Introducing system one models & Jev. https://typesafe.ai/blog/introducing-system-one-models-and-jev, September 2026. TypeSafe AI.

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. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding, 2026a. URL https://arxiv.org/abs/2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shirui Chen, Jiantao Jiao, Lillian J Ratliff, and Banghua Zhu. dultra: Ultra-fast diffusion language models via reinforcement learning. *arXiv preprint arXiv:2512.21446*, 2025.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. Dmax: Aggressive parallel decoding for dllms. *arXiv preprint arXiv:2604.08302*, 2026b.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dllms. In *International Conference on Learning Representations*, volume 2026, pp. 53193–53211, 2026c.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

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

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Phuong Tuan Dat, Qi Li, and Xinchao Wang. dopsd: On-policy self-distillation for diffusion language models. *arXiv preprint arXiv:2607.04428*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, et al. Diffusiongemma technical report. *arXiv preprint arXiv:2608.00146*, 2026.

Charles Frye and Shankha Biswas. Achieve state-of-the-art inference latencies with speculative decoding. https://modal.com/blog/achieve-sota-specdec, June 2026. June 24, 2026.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025a.

Jonas Geiping, Xinyu Yang, and Guinan Su. Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models. *arXiv preprint arXiv:2510.14961*, 2025b.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Yanzhe Hu, Yijie Jin, Pengfei Liu, Kai Yu, and Zhijie Deng. Lightningrl: Breaking the accuracy-parallelism trade-off of block-wise dllms via reinforcement learning. *arXiv preprint arXiv:2603.13319*, 2026a.

Yuezhou Hu, Harman Singh, Monishwaran Maheswaran, Haocheng Xi, Coleman Hooper, Jintao Zhang, Aditya Tomar, Michael W Mahoney, Sewon Min, Mehrdad Farajtabar, et al. Residual context diffusion language models. *arXiv preprint arXiv:2601.22954*, 2026b.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026.

Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. *arXiv preprint arXiv:2212.11972*, 2022.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. *arXiv preprint arXiv:2602.11451*, 2026.

Haozhe Jiang, Haiwen Feng, Pieter Abbeel, Jiantao Jiao, Angjoo Kanazawa, and Nika Haghtalab. Diffusion policy optimization without drifting apart. *arXiv preprint arXiv:2606.13795*, 2026.

Renren Jin, Pengzhi Gao, Yuqi Ren, Zhuowen Han, Tongxuan Zhang, Wuwei Huang, Wei Liu, Jian Luan, and Deyi Xiong. Revisiting entropy in reinforcement learning for large reasoning models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25300–25322, 2026.

Mingyu Jo, Jaesik Yoon, Justin Deschenaux, Caglar Gulcehre, and Sungjin Ahn. Loopholing discrete diffusion: Deterministic bypass of the sampling wall. In *International Conference on Learning Representations*, volume 2026, pp. 93890–93918, 2026.

Marius Killinger and Philip Kiely. Why GPU utilization matters for model inference. https://www.baseten.co/blog/why-gpu-utilization-matters-for-model-inference/, 2025. Baseten, last updated May 18, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Sanghyun Lee, Chunsan Hong, Seungryong Kim, Jonghyun Lee, Jongho Park, and Dongmin Park. Looped diffusion language models. *arXiv preprint arXiv:2605.26106*, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xiaocheng Lu, Hualei Zhang, Shuhan Guo, Jie Zhang, Xiaoyi Pang, Jian Liu, Haoxi Li, Bohai Gu, Haoxuan Che, Jingcai Guo, et al. Optd: On-policy transition distillation with consistency-guided adaptive compression for few-step diffusion language models. *arXiv preprint arXiv:2608.02942*, 2026.

Yifu Luo, Zeyu Chen, Haoyu Wang, Xinhao Hu, Yuxuan Zhang, Zhizhou Sha, and Shiwei Liu. Learning from the self-future: On-policy self-distillation for dllms. *arXiv preprint arXiv:2606.18195*, 2026.

Nebius team. AI model components: Key elements of a GenAI inference setup explained. https://nebius.com/blog/posts/ai-model-components-key-elements-of-a-genai-inference-setup-explained, August 2025. August 5, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. d3llm: Ultra-fast diffusion llm using pseudo-trajectory distillation. *arXiv preprint arXiv:2601.07568*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*, 2024.

Yuji Ren, Chenkai Xu, Zhuocheng Gong, Jianguo Li, and Zhijie Deng. Cforce: Boosting parallel decoding for dllms via consistency forcing. *arXiv preprint arXiv:2608.13925*, 2026.

Shane Robbins. Production AI runs on inference. are you ready for it? https://www.coreweave.com/blog/production-ai-runs-on-inference-are-you-ready-for-it, June 2026. CoreWeave.

Benjamin Rozonoyer, Jacopo Minniti, Dhruvesh Patel, Neil Band, Avishek Joey Bose, Tim GJ Rudner, and Andrew McCallum. Learned relay representations for forward-thinking discrete diffusion models. *arXiv preprint arXiv:2605.22967*, 2026.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, Kai Yu, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. In *International Conference on Learning Representations*, volume 2026, pp. 2282–2299, 2026a.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. In *International Conference on Learning Representations*, volume 2026, pp. 101476–101501, 2026b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. In *International Conference on Learning Representations*, volume 2026, pp. 128353–128370, 2026.

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.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, Tri Dao, Ben Athiwaratkun, James Zou, Fan Lai, and Chenfeng Xu. Introspective diffusion language models. *arXiv preprint arXiv:2604.11035*, 2026.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:56729–56762, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Related Works

Residual diffusion language models. Retaining continuous information across denoising steps lets a DLM reuse computation beyond its discrete token commitments, building on self-conditioning in continuous diffusion (Chen et al., 2022; Jabri et al., 2022). Loopholing (Jo et al., 2026) introduces a deterministic latent pathway that bypasses the information loss of categorical sampling; RCD (Hu et al., 2026b) constructs entropy-weighted residual embeddings from predictive distributions and trains the model with reference-generated residual signals; and DiffusionGemma (DiffusionGemma Team et al., 2026) incorporates distribution-based self-conditioning into blockwise generation. Closely related, Relay (Rozonoyer et al., 2026) trains a cross-step hidden-state channel through truncated backpropagation through time, committing ground-truth token values during its training rollouts. These approaches establish the value of cross-step feedback, but their training states are typically constructed from ground-truth sequences or reference-generated signals. We identify a further challenge: as the model changes, both its residual predictions and the trajectories they induce change together. Our multi-seed comparison of every-step and infrequent rollout refresh shows distinct learning curves (Figure 2(a)); protocol details and result aggregation are reported in Appendix H.4. To track the evolving distribution, ORL adopts a fully on-policy framework: every update uses newly generated trajectories from the current model, retaining verified responses and replaying intermediate states with the exact residual inputs consumed during generation. Adjacent entropy contraction encourages faster refinement, while our co-designed training system makes this workflow practical at scale.

Reinforcement learning and on-policy post-training for DLMs. d1 (Zhao et al., 2025) adapts policy-gradient optimization to masked DLMs, while DiPOD (Jiang et al., 2026) introduces on-policy ELBO regularization to mitigate drift between its ELBO surrogate and the true log-likelihood. dUltra (Chen et al., 2025) and LightningRL (Hu et al., 2026a) target decoding efficiency through learned unmasking decisions and efficiency-aware rewards. However, generating responses on-policy does not by itself ensure that training preserves the states encountered during decoding: LightningRL's released implementation constructs block-masked training inputs from completed responses rather than replaying the intermediate decoding states. TraceRL (Wang et al., 2026b) aligns training with the decoding trace, preserving step-level conditioning when no trajectory shrinkage is applied, but its policy objective supervises the token actions committed at each step rather than predictions at positions that remain unresolved. Related non-RL approaches also exploit model-generated trajectories: d-OPSD (Luo et al., 2026) constructs a self-teacher by selectively revealing tokens from generated completions, dOPSD (Dat et al., 2026) distills later-state predictions into earlier states, and OPTD (Lu et al., 2026) learns compressed transitions on student-visited states. ORL combines exact residual-state replay with supervision beyond the sampled actions. At selected states of verified rollouts, it supervises intermediate predictions toward the final response, including at positions not committed at that step. Adjacent entropy contraction further encourages uncertainty reduction over the unresolved canvas between consecutive steps, providing a differentiable signal for denoising progress rather than relying solely on trajectory-level efficiency rewards. Both objectives operate within a fully on-policy framework that regenerates trajectories after every update and preserves the exact residual context consumed at each training state.

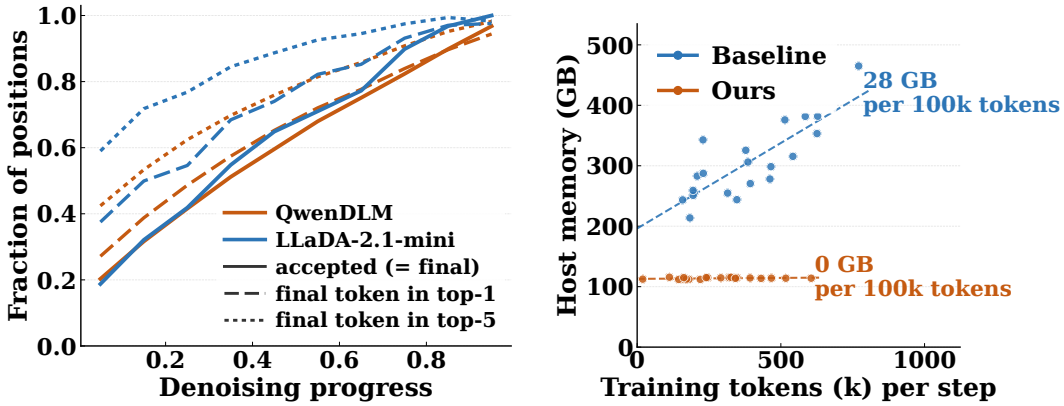
Looped Transformers and latent computation. Recurrent computation offers a way to increase reasoning depth without proportionally increasing model parameters. Universal Transformers (De-

hghani et al., 2018) introduce shared transformations with adaptive computation, and subsequent work studies their reasoning benefits (Saunshi et al., 2025) and scales latent recurrence to language modeling (Geiping et al., 2025a; Zhu et al., 2025). LoopFormer (Jeddi et al., 2026) further uses shortcut-consistency training to support variable computation budgets. This perspective connects naturally to diffusion generation: Geiping et al. (2025b) connect recurrent-depth models to diffusion through parallel sampling, while LoopMDM (Lee et al., 2026) introduces selective layer recurrence within masked diffusion models. Residual DLMs realize recurrent computation through continuous predictive feedback between denoising steps, without requiring additional layer repetitions within each forward pass. ORL is complementary to these architectural advances: it trains the cross-step feedback on fresh trajectories from the current model, with the exact residual context encountered during generation, and combines verified-output supervision with an explicit refinement objective so that intermediate predictions support accurate outputs with fewer decoding steps.

B Residual Diagnostics

B.1 Predictive Information and Capture Cost

Intermediate predictions often contain the final decoded token before it is committed (Figure 7a). This is agreement with the trajectory's output, not reference-answer correctness. Retaining such states also creates a memory bottleneck: naive capture adds about 28 GB of host memory per 100k training tokens, versus 36 GB for the combined host/GPU footprint in Figure 2(c).



(a) Intermediate top-k agreement with final tokens.

(b) Host memory versus materialized training tokens.

Figure 7: Residual-learning motivation. Left: top-1/top-5 agreement exceeds the accepted-and-final fraction early in denoising. Right: sparse, GPU-resident capture avoids the baseline's growing host footprint over the measured range.

B.2 How On-Policy Training Changes Model Behavior

Figure 8 (orange) tracks the entropy of the rollout policy over training: it decreases steadily from 0.20 to 0.14 rather than collapsing, matching the graded increase in TPF. Figure 8 (blue) tracks the covariance between the log-probability of sampled tokens and their advantage-weighted probability (Appendix I.3), the quantity that sets the direction of entropy change under a policy-gradient update. It stays positive on average, so successive updates sharpen the distribution, and it shrinks toward zero as training proceeds, so sharpening slows down instead of running away (Cui et al., 2025; Jin et al., 2026). Together with the residual-to-embedding norm ratio, which rises from 0.91 to 1.08 during on-policy training (Figure 17), and the bounded residual-pathway update ratios in Appendix H.2, this indicates that ORL changes how intermediate predictions are refined across denoising steps rather than merely improving the final-token predictor, and that it does so by making greater use of residual context without destabilizing the residual pathway.

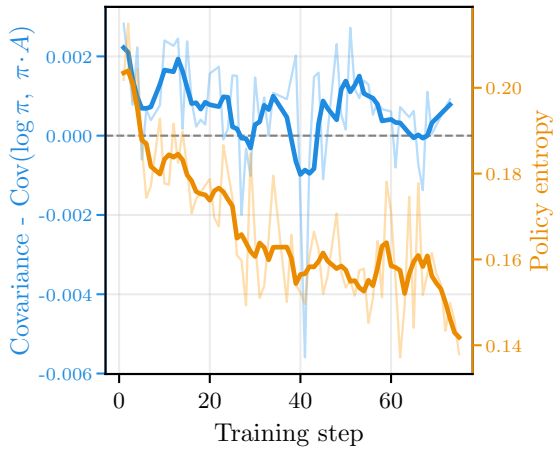


Figure 8: Dynamics of on-policy residual learning. The entropy-change covariance (blue, left axis) remains positive but approaches zero, while rollout entropy (orange, right axis) decreases gradually without collapse.

C Optimization Details and Stabilization

Response normalization. For verified responses $\mathcal{R}_{\mathcal{B}}^{+}$, retained states $\mathcal{S}(\mathcal{R})$, and valid supervised positions $\mathcal{I}_b^{\mathcal{R}}$ (excluding padding and positions after terminal EOS), let $N_{\mathcal{R}} = \sum_{(b,k)\in\mathcal{S}(\mathcal{R})} |\mathcal{I}_b^{\mathcal{R}}|$. QwenDLM and LLaDA use

$$\mathcal{A}[\ell] = \frac{1}{|\mathcal{R}_{\mathcal{B}}^{+}|} \sum_{\mathcal{R}\in\mathcal{R}_{\mathcal{B}}^{+}} \frac{1}{N_{\mathcal{R}}} \sum_{(b,k)\in\mathcal{S}(\mathcal{R})} \sum_{i\in\mathcal{I}_b^{\mathcal{R}}} \ell_{\mathcal{R},b,k,i}. \tag{3}$$

Thus each verified response has equal token-loss weight; empty verified batches are skipped.

AEC aggregation. A valid pair $(b,k) \in \mathcal{P}(\mathcal{R})$ has a selected target state $(b,k+1)$ and nonempty unresolved sets at both states. The source need not be selected: its captured mean normalized entropy supplies the reference. Give each pair weight $w_{\mathcal{R}} = 1/|\mathcal{P}(\mathcal{R})|$ before gating. QwenDLM uses $g_{\mathcal{R},b,k} = \mathbf{1}\{H_{b,k}^{\text{roll}} \le 0.20\}$; LLaDA uses $g = 1$. With $D = \sum_{\mathcal{R},(b,k)\in\mathcal{P}(\mathcal{R})} w_{\mathcal{R}} g_{\mathcal{R},b,k}$,

$$\mathcal{L}_{\text{AEC}} = \frac{\sum_{\mathcal{R},(b,k)\in\mathcal{P}(\mathcal{R})} w_{\mathcal{R}} g_{\mathcal{R},b,k} \ell_{\mathcal{R},b,k}^{\text{AEC}}}{D}, \qquad D > 0, \tag{4}$$

and the loss is zero for $D = 0$. Responses with no valid pair are excluded. Weights are not recomputed after gating: a response's remaining mass is its retained fraction of pairs. DiffusionGemma instead uses the canvas-pair mean in Appendix E.4.

Entropy-gated supervision. For detached reference q_i^{cap}, replayed distribution $\widehat{p}_i$, normalized entropy $\overline{H}$, and cap C,

$$\ell_i^{\text{hybrid}} = \begin{cases} -\log \widehat{p}_i(y_i), & \overline{H}(q_i^{\text{cap}}) > \zeta, \\ \min\{D_{\text{KL}}(\widehat{p}_i \| q_i^{\text{cap}}), C\}, & \text{otherwise.} \end{cases} \tag{5}$$

Only QwenDLM's final training stage uses this gate, with $\zeta = 0.10$ and $C = 5$; all other reported training stages use CE. The teacher is reconstructed from each state's captured, shifted output-head-input hidden vector using the *current* output head under stop-gradient, not an old head or one final-response teacher. Both head outputs are cast to float32, divided by loss temperature 1, clipped to $[-50, 50]$, then normalized over the vocabulary. Replay gradients accumulate before one optimizer step.

For $p = \text{softmax}(a)$ and $D_{\text{KL}}(p\|q) = D$, $\partial D/\partial a_j = p_j[\log(p_j/q_j) - D]$ before capping. Identical transformed distributions therefore give zero RKL loss and gradient. Exact captured inputs do not guarantee rollout/replay numerical parity; any nonzero RKL contribution in this training recipe was not measured separately from the effect of removing low-entropy CE.

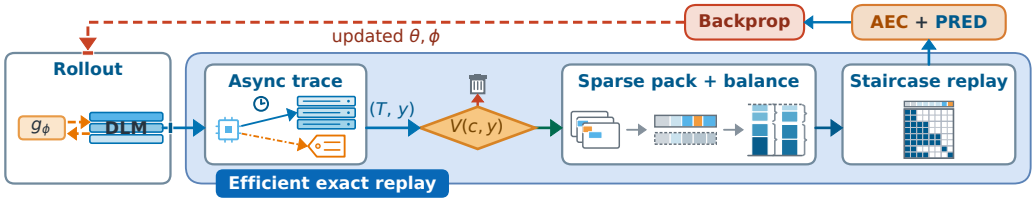


Figure 9: **Overview of ORL.** The current model generates residual-conditioned trajectories with exact state capture; verified states are packed and load-balanced across GPUs, replayed with staircase attention, and optimized using CE and AEC.

Residual initialization. QwenDLM's bias-free adapter is $g_\phi(z) = W_d[\mathrm{SiLU}(W_g u) \odot W_u u]$, $u = \mathrm{RMSNorm}_w(z)$, with hidden/intermediate dimensions 2,560/9,728 and epsilon 10^{-6}. Multiplying all three matrices by $\gamma = 8$ and dividing both the normalized input and branch output by γ preserves its function:

$$\widetilde{g}(z) = \frac{\gamma W_d}{\gamma}[\mathrm{SiLU}(\gamma W_g u/\gamma) \odot (\gamma W_u u/\gamma)] = g_\phi(z). \tag{6}$$

The RMSNorm gain is unchanged. Both compensation constants are $1/8$, distinct from injection strength $\alpha = 1$; the residual sum then receives non-affine RMS normalization scaled by 0.02. The same post-normalization is applied during replay to z in Equation 1. Compensation is applied once, before initializing the ORL optimizer, not again on resume. Output-only compensation would not preserve SiLU's arguments.

LLaDA uses affine pre-RMSNorm and three bias-free gated-MLP projections; activation, dimensions, and normalization epsilon follow the model config. It initializes gate/up weights from a Gaussian with standard deviation 10^{-3}, down weights to zero, and pre-RMSNorm gain to one. All adapter parameters use 0.3 times the backbone learning rate. For LLaDA, self-conditioning probability and injection strength are both fixed at one; no injection ramp or post-normalization is used. Initialization is checked after meta-materialization and before FSDP synchronization. DiffusionGemma retains its native self-conditioning module.

D System Implementation Details

Figure 9 summarizes the ORL loop and where each mechanism below acts.

D.1 Asynchronous Exact-State Capture

Each forward snapshots mutable token IDs and accept/edit masks, retains the residual and required GPU statistics, records a CUDA event, and enqueues a bounded handle. A background worker materializes the record after the event; the decode critical path performs no host copy or host synchronization. Exhausting the queue's entry/byte budget blocks for a bounded interval or fails explicitly, never silently dropping states. Host-bound fields use pinned memory. With a colocated trainer, residual/hidden-source pages can remain GPU-resident and cross through one-shot CUDA-IPC handles, relaying only required pages to another GPU. Storage remains live until consumer acknowledgement.

D.2 Staircase Attention

A noisy block attends to the prompt, preceding clean blocks, and itself, but not its clean target, later blocks, other noisy blocks, or other documents. Clean queries never attend to noisy keys. The Triton kernel derives visible key ranges from per-document metadata, skips invisible tiles, and applies the predicate within boundary tiles; it allocates no dense $N \times N$ mask. Backward uses the same visibility: dQ follows the query traversal, while dK, dV derive valid query ranges per key tile. The implementation supports variable-length packed documents, grouped-query attention, and sequence parallelism.

D.3 Packing and Cost Balancing

Same-round states share clean context, following the slicing structure of TraceRL (Wang et al., 2026b); residual/source pages remain sparse and block-local. Replay rows are packed as mutually invisible documents. A weighted proxy of sequence tokens, supervised positions, and resident pages drives deterministic largest-cost-first assignment to data-parallel groups. Chunked materialization bounds the live input/page footprint. Under hybrid underfill, splittable label sets are partitioned before zero-loss ghost rows align collective schedules. Each label has one owner, and global reduction uses $\mathcal{L} = \sum_j \mathrm{num}_j / \sum_j \mathrm{den}_j$, preserving the unpacked objective up to rounding.

E Training and Evaluation Details

E.1 Evaluation Protocols

Measured throughput uses one H100, TP=1, batch size one, and one request in flight. Length-terminated responses count as incorrect and are excluded only from clean efficiency statistics. Request time includes prefill, decoding, scheduling, and host overhead, but excludes model loading, server startup, and preprocessing. Warmup exclusions are stated per run. Within each family, Base and ORL share tasks and scorers. Each evaluation record is associated with its checkpoint and evaluation profile in the anonymous repository (Appendix E.1). Runtime acceleration is evaluated separately in Appendix F.

Runtime settings. QwenDLM and LLaDA use SGLang (Zheng et al., 2024); DiffusionGemma uses vLLM (Kwon et al., 2023) in BF16. Engine revisions, container digests, attention backends, cache controls, and graph settings are recorded in the artifact's `configs/paper/decoding/`. Absolute TPS is hardware/runtime-specific; the artifact's separate small same-GPU check does not replace the paper's full-task timing runs.

Tasks and scoring. The main suite uses GSM8K test (1,319 tasks) (Cobbe et al., 2021), MATH-500 (500) (Hendrycks et al., 2021), HumanEval (164, base tests) (Chen et al., 2021), and the canonical MBPP427 collection (Austin et al., 2021) with its original multi-candidate extraction. Base/ORL share task and scorer protocols within each family; cross-family and literature comparisons are not scorer-matched. QwenDLM retains its CodeTACO code-scoring path, while LLaDA/DiffusionGemma use EvalPlus base tests for HumanEval. The artifact's `repro/paper/main.yaml` and `repro/eval/portable/protocols.json` record input identities, native prompting, extraction rules, and scorer pins.

Runs and aggregation. For each ORL configuration, we conduct multiple independent training runs and repeated evaluations of each resulting checkpoint. In the main benchmark suite, the same checkpoint from each training run is evaluated across all benchmarks. We first average the evaluation metrics within each checkpoint, then average equally across independent training runs. This averaging protocol also applies to the component and hyperparameter ablations in Table 3. Starting-model baselines average repeated evaluations of their reference checkpoints. Each evaluation generates one completion per task; DiffusionGemma uses three generation-seed settings per checkpoint. Seed settings and task partitions are recorded in `repro/paper/main.yaml`; transfer-evaluation settings are given below. Reported means alone do not establish statistical significance for small score differences. Within an individual HumanEval evaluation, one task corresponds to approximately 0.61 percentage points; averaging repeated evaluations permits smaller increments in the reported means.

Reproduction configurations. For the complete generation configuration associated with each reported checkpoint and operating point, refer to our anonymous repository. The experiment index at `configs/paper/experiments.yaml` maps results to their profiles in `configs/paper/decoding/`; `docs/reproduction/SETTINGS.md` provides the configuration guide. Profile identifiers in the tables below refer to these records.

Table 4: Transfer evaluation. Evaluation-seed counts are generation-seed settings per trained checkpoint (LL: LLaDA; DG: DiffusionGemma). ORL metrics average independent training runs and repeated evaluations; starting-model baselines average repeated evaluations of their reference checkpoints. Each evaluation generates one completion per task. AIME-24 scores 30 tasks from a shared 60-task, two-year generation input.

Benchmark	Tasks	Eval. seeds LL/DG
AIME-24	30	8 / 8
GPQA-Diamond	198	3 / 3
MMLU-Pro	12,032	1 / 3
LiveCodeBench-v6	175	1 / 3
IFEval	541	1 / 3

Accounting. Let $\mathcal{I}_b$ be all requests, $\mathcal{C}_b$ those not length-terminated, s_i the scorer outcome, C_i completion tokens, D_i denoising forwards, and W_i complete request time. We report

$$\mathrm{Acc}_b = \frac{100}{|\mathcal{I}_b|}\sum_{i\in\mathcal{C}_b} s_i, \qquad \mathrm{TPF}_b = \frac{1}{|\mathcal{C}_b|}\sum_{i\in\mathcal{C}_b}\frac{C_i}{D_i}, \tag{7}$$

$$\mathrm{TPS}_b^{\mathrm{clean}} = \frac{\sum_{\mathcal{C}_b} C_i}{\sum_{\mathcal{C}_b} W_i}, \qquad \mathrm{TPS}_b^{\mathrm{all}} = \frac{\sum_{\mathcal{I}_b} C_i}{\sum_{\mathcal{I}_b} W_i}, \tag{8}$$

$$\mathrm{TPS}_b^{\mathrm{completed}} = \frac{\sum_{\mathcal{C}_b} C_i}{\sum_{\mathcal{I}_b} W_i}. \tag{9}$$

Completed-output TPS charges failed-request time without crediting its tokens, but still credits completed incorrect answers; it is not correct-answer throughput. All-forward TPF replaces D_i by $F_i = D_i + R_i + K_i$ (denoise, refinement, commit). Wall TPS includes all this work; the approximation $\mathrm{TPF}/t_{\mathrm{forward}}$ does not. We compute benchmark metrics for each evaluation and average repeated evaluations within each checkpoint. For ORL, we then average equally across independent training runs. Cross-benchmark summaries take an equal-weight mean of the benchmark-level metrics.

Baseline sources. The underlined starting points and ORL rows in Table 1 are ours; other rows are paper-reported. Fast-dLLM–LLaDA (Wu et al., 2025), dParallel–LLaDA (Chen et al., 2026c), D2F–LLaDA (Wang et al., 2026a), Fast-dLLM-v2-7B (Wu et al., 2026), and d3LLM follow the unified d3LLM evaluation (Qian et al., 2026), whose MATH entries are MATH (4-shot) on the full test set rather than MATH-500; SDAR-8B-b32 (Cheng et al., 2025) follows LightningRL (Hu et al., 2026a). RCD (Hu et al., 2026b), dUltra-coding (Chen et al., 2025), and DMax-Math (Chen et al., 2026b) use their original papers. TraDo-4B (dynamic) is the TraceRL-trained model reported by Wang et al. (2026b). For dOPSD (Dat et al., 2026), we quote the LLaDA-8B-Instruct results from Table 1: GSM8K/MATH-500 accuracy $72.87/36.00$ and HumanEval/MBPP pass@1 $39.63/39.54$, with math/code averages $54.44/39.59$. Corresponding TPF is unreported and marked with dashes. These published comparisons retain their original backbones and evaluation settings; they are not scorer-matched reruns.

Derived comparison points are estimates, not measured wall TPS. Their conversion is $\widehat{\mathrm{TPS}}_{m,b} = c_b\mathrm{TPF}_{m,b}$, with c_b taken from a measured reference's TPS/TPF or the mean of two matched endpoint ratios. Disaggregating a paper's aggregate TPS instead uses $\widehat{\mathrm{TPS}}_{m,b} = \mathrm{TPS}_m^{\mathrm{paper}}\mathrm{TPF}_{m,b}/\overline{\mathrm{TPF}}_m$ under that paper's task, hardware, and decoding protocol.

E.2 Additional Benchmarks

Table 4 summarizes the transfer suite: AIME-24 (AIME, 2024), GPQA-Diamond (Rein et al., 2024), MMLU-Pro (Wang et al., 2024), LiveCodeBench-v6 (Jain et al., 2024), and IFEval (Zhou et al., 2023). The artifact's `repro/paper/transfer.yaml` and `repro/paper/TRANSFER_GENERATION.md` record exact seeds, input hashes, prompt serialization, context settings, and scorer profiles.

Quality and clean per-response TPF are first averaged over repeated evaluations of each checkpoint; ORL metrics are then averaged equally across independent training runs. Quality is not best-of-k or majority voting. AIME uses numeric parsing, GPQA/MMLU-Pro the original multiple-choice parser, LiveCodeBench execution tests, and IFEval its instruction evaluator. Historical source/RNG binding for the LCB/IFEval reproduction scorers and the DiffusionGemma long-task context override remains incomplete; the artifact records these separately from fresh-run settings.

Transfer quality. In Table 2, TPF increases on all ten model–benchmark pairs, while accuracy changes vary by task. LLaDA's accuracy declines by 3.4, 1.8, and 1.7 percentage points on GPQA-Diamond, IFEval, and AIME-24, respectively; DiffusionGemma's accuracy declines by 2.1 percentage points on LiveCodeBench-v6.

E.3 Offline SDAR Initialization

QwenDLM starts from a non-residual Qwen3-4B-Thinking-2507 SDAR checkpoint. We add a zero-output residual adapter and train with CE on ReasonMix/SFTMix, using block size 256 and an 8,192-token cutoff. The initialization checkpoint for the on-policy experiments and the full offline recipe, including optimizer, normalization, and hardware settings, are recorded in `configs/paper/training.yaml`. An auxiliary sweep evaluated 118 checkpoints on GSM8K100 and HumanEval164; later checkpoints did not monotonically improve accuracy.

E.4 On-Policy Training Recipes

We train each model until tokens per forward (TPF) stabilizes. Each iteration draws one prompt and eight rollouts and updates on verified responses only; batches with no verified response do not trigger an optimizer update. The artifact's `configs/paper/training.yaml` records the complete per-model recipes, training schedules, checkpoint identities, rollout controls, adapter initialization, optimizer settings, resume records, and compute accounting.

Data. QwenDLM uses 819 OpenCodeInstruct-derived Python problems. LLaDA and DiffusionGemma use a balanced mixture of those code problems, 819 GSM8K problems, and 819 MATH problems. Code rewards use the supplied test suite; math rewards use task-specific verifiers. Exact normalized checks found no final-training-file matches against the corresponding HumanEval, GSM8K, and MATH-500 evaluation sets; semantic and earlier-stage contamination are not established by this check.

QwenDLM. After residual rescaling by $\gamma = 8$, we train at learning rate 10^{-5}, then continue at 5×10^{-6} with the entropy-gated CE/RKL loss. Response-normalized CE is combined with AEC (coefficient 50, $\rho = 0.05$, source-entropy ceiling 0.20), using the state selection in Section 3.

LLaDA-2.1-mini. We add the zero-output adapter and train with block size 32, backbone/adapter learning rates $10^{-5}/3 \times 10^{-6}$, and response-normalized CE plus AEC (coefficient 50, $\rho = 0.20$), without entropy-gated RKL. Length-truncated responses are rejected.

DiffusionGemma. We retain native self-conditioning and train with a 256-token canvas, keeping every intermediate state. Gradients pass through the current canvas forward; captured residuals and prefix-cache construction are detached. Token-mean CE is combined with the mean over valid adjacent canvas pairs within the same block:

$$\mathcal{L}_{\mathrm{AEC}}^{\mathrm{DG}} = \frac{1}{|\mathcal{P}_{\mathrm{DG}}|} \sum_{(t,t+1)\in\mathcal{P}_{\mathrm{DG}}} [H_{t+1}^{\theta} - 0.8\ \mathrm{sg}(H_t^{\theta})]_+^2. \tag{10}$$

Both entropies are recomputed by the actor and the source is detached. The AEC coefficient is 100, with no entropy gate. AdamW uses learning rate 10^{-5}. Training uses an eight-GPU actor and a separate eight-GPU rollout worker.

F Inference Optimization

We reduce inference overhead through cache reuse, fused execution, and CUDA graphs. Implementation details and configuration-specific compatibility checks are provided in the anonymous repository (Appendix E.1). The final output-parity gate covers HE164, EvalPlus MBPP378, and GSM8K100.

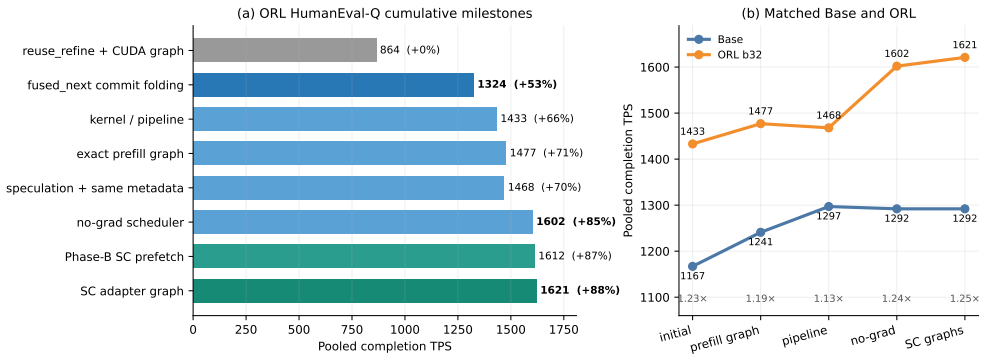


Figure 10: Cumulative LLaDA inference optimizations, HumanEval Quality mode. Each side uses four one-H100 engines, pooled completion TPS, and removes the first warmup request per shard. The ORL checkpoint differs from the main-table checkpoint. The 864-to-1,324 transition had 153/164 identical outputs and accuracy $0.896 \rightarrow 0.890$, not exact parity. Later changes pass the final gate. The last two normalized gains are 0.60%/1.18%; their raw paired comparison is $1610.2 \rightarrow 1629.2$ TPS.

Cost and scope. Before the later graph fixes, ORL improves TPF by $1.51\times$ but TPS by $1.23\times$: Base/ORL forwards take 7.47/9.17 ms. After optimization, they take 6.74/8.11 ms. Self-conditioning adds work and fewer forwards amortize fixed costs less well, so TPF is not a throughput substitute. Supported execution modes and compatibility requirements are specified in the repository. Output parity is rechecked on new hardware.

G Throughput and Additional Comparisons

G.1 Full-Task Request Accounting

Table 5 covers the same GSM1319/HE164 IDs on both sides, with no warmup exclusion. Qwen problem text, gold, and prompt-token counts match; wire prompts were not archived. LLaDA lacks prompt text and HE gold tests, so task-ID agreement alone does not establish prompt/test parity.

QwenDLM's benchmark-macro clean TPS is 186.88 for Base and 933.30 for ORL, giving approximately $5\times$ as reported in the main text. All-output/completed-output TPS ratios are $4.59/4.70$. At LLaDA's main-table operating points, the clean/all-output/completed-output ratios are $1.23/1.25/1.29$. These values follow the request-accounting definitions in Appendix E.1. The approximately $3\times$ LLaDA result corresponds to the pair with the same observed mean accuracy in Figure 12 ($2.95\times$ clean TPS). Qwen Base has one numerically correct but length-terminated GSM answer; counting it as a failure gives $1191/1319 = 90.30\%$ and mean GSM/HE accuracy 86.61% versus 91.03%.

G.2 Speculative-Decoding Comparison

Table 6 reconstructs the GSM8K1319/HumanEval164 comparison in Figure 1b from per-benchmark summaries. Accuracy and TPS each average the two benchmark-level metrics. Within each benchmark, TPS pools tokens and request time over each setting's own clean requests (Equation 8), not a common-clean intersection across settings.

Table 5: Original full-task main runs. L/N counts length failures over all tasks. Lengths are mean returned tokens on clean/all requests. TPS columns use Equations 8 and 9: clean, all-output, and completed-output/all-request-time, respectively. Quality always includes all tasks, with length failures incorrect.

Role	Task	Acc. (%)	L/N	Length clean/all	Clean TPS	All TPS	Completed TPS
QwenDLM							
Base	GSM8K	90.30	1/1319	1723/1759	209.7	213.5	209.0
ORL	GSM8K	92.42	1/1319	1140/1164	718.4	724.8	709.3
Base	HumanEval	82.93	6/164	4929/5948	164.1	196.0	156.5
ORL	HumanEval	89.63	4/164	5621/6283	1148.2	1153.7	1007.0
LLaDA-2.1-mini							
Base	GSM8K	91.89	11/1319	265/297	715.7	669.1	592.1
ORL	GSM8K	92.04	2/1319	216/222	949.2	938.7	912.5
Base	HumanEval	86.59	0/164	360/360	1302.3	1302.3	1302.3
ORL	HumanEval	89.02	0/164	546/546	1530.1	1530.1	1530.1

Table 6: Speculative-decoding results on GSM8K1319 and HumanEval164 (Figure 1b). Accuracy and clean TPS each average the two benchmark-level metrics. Length failures count as incorrect and are excluded from clean TPS; counts cover both tasks. Profile definitions are provided in the anonymous repository (Appendix E.1). High-failure profiles remain visible, but clean TPS does not charge their failed-request time.

Target / method	Profile	Mean acc. (%)	Clean TPS	Length failures
Qwen / AR	QA	93.40	240.1	2
Qwen / EAGLE3	QE01	93.47	352.7	3
	QE02	94.23	404.7	3
	QE03	88.39	432.3	23
	QE04	81.08	462.6	69
	QE05	65.58	483.4	220
	QE06	48.63	467.6	456
Qwen / DFlash	QD01	93.44	573.3	3
	QD02	92.45	690.9	3
	QD03	90.76	745.5	12
	QD04	87.15	788.6	32
	QD05	80.37	827.1	75
Gemma / AR	GA	96.76	206.8	2
Gemma / DFlash	GD01	96.80	549.1	1
	GD02	97.47	605.9	2
	GD03	96.83	634.0	3
	GD04	95.77	639.8	10
	GD05	94.20	662.7	21
Gemma / EAGLE3	GE01	96.83	331.1	0
	GE02	96.91	351.3	6
	GE03	94.72	362.4	13
	GE04	90.43	367.0	44
	GE05	84.45	385.4	111

For the Qwen comparison, matched quality uses the main text's one-sided criterion: the fastest measured setting per drafter with mean accuracy at least ORL's minus 0.5 percentage points. At ORL's 91.03%, the corresponding profiles are EAGLE3 QE02 (94.23%, 404.7 TPS) and DFlash QD03 (90.76%, 745.5 TPS); accuracies need not be equal. Speculative TPF is $1 + N_{\text{accepted}}/N_{\text{draft}}$, not denoising TPF.

Configuration. Each server uses one H100, vLLM, and batch size one; two warmup requests precede measurement. The anonymous repository records the complete configuration for each profile in Table 6 (Appendix E.1), including target and drafter checkpoint identities. The Qwen DFlash drafter targets a different Qwen release.

G.3 Additional Model Comparisons

Figure 11 extends the GSM8K/HumanEval comparison in Figure 1b with additional model families and methods, including TraDo-8B (Wang et al., 2026b). CForce (Ren et al., 2026) and Nemotron-Diff (Fu et al., 2026) use published results converted to TPS; LightningRL (Hu et al., 2026a) uses our measured TPF converted to TPS (Appendix E.1). These converted points indicate approximate throughput positions. The remaining points use our measured wall-clock completion throughput on one H100 at batch size one.

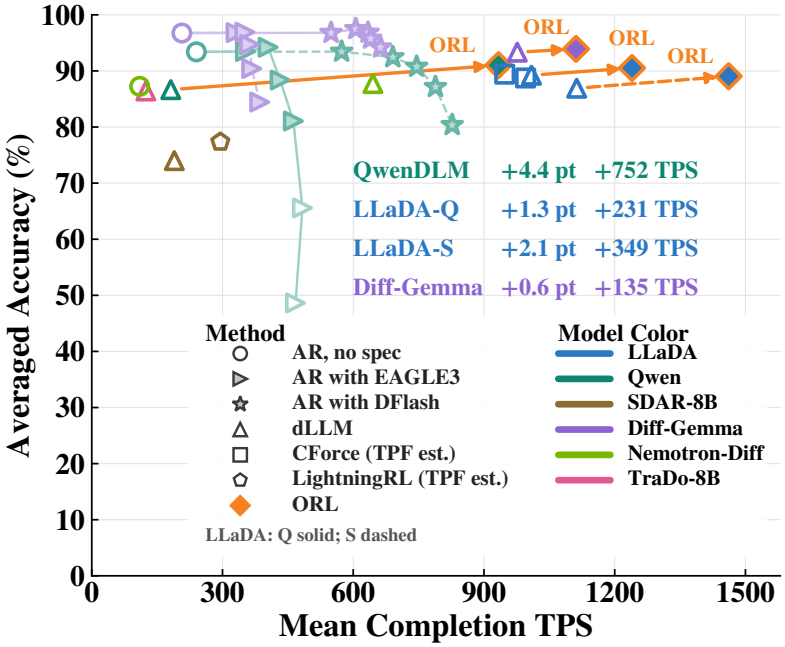


Figure 11: **Accuracy–throughput on GSM8K and HumanEval with additional baselines.** Both axes average the two benchmark-level metrics. Marker shape denotes the method and color the model family; orange arrows connect each Base DLM to its ORL model. LLaDA-Q and LLaDA-S denote Quality and Speed modes. TPS for CForce, LightningRL, and Nemotron-Diff is estimated as described above.

G.4 Quality–Throughput Sweeps

Figure 12 contains all 48 operating points: ten each for Qwen Base and ORL, and fourteen per LLaDA checkpoint. Qwen uses common GSM IDs 0–399 and HE164; LLaDA uses GSM1319 and HE164. The complete configuration of each point is indexed in the anonymous repository (Appendix E.1).

G.5 Accuracy–Canvas-TPF Tradeoffs

For K_i committed blocks of size $B = 32$, define

$$\mathrm{TPF}_b^{\mathrm{canvas}} = \frac{1}{|\mathcal{C}_b|} \sum_{i \in \mathcal{C}_b} \frac{BK_i}{D_i}. \tag{11}$$

Unlike completion TPF, it includes final-block padding; it does not subtract edits or charge refinement/commit forwards. It is a refinement diagnostic, not wall TPS. The locked Quality sweep uses the same tasks at all fourteen settings, separately for HumanEval and GSM8K. At the highlighted HumanEval pair, both checkpoints pass 146/164 tasks; canvas TPF rises from 4.8395 to 13.9086 (2.874×), while completion TPF rises from 4.3668 to 13.0771 (2.995×). Seven tasks pass only each side; accuracy matching is empirical. Appendix J.2 gives comparisons at shared settings and curve validation.

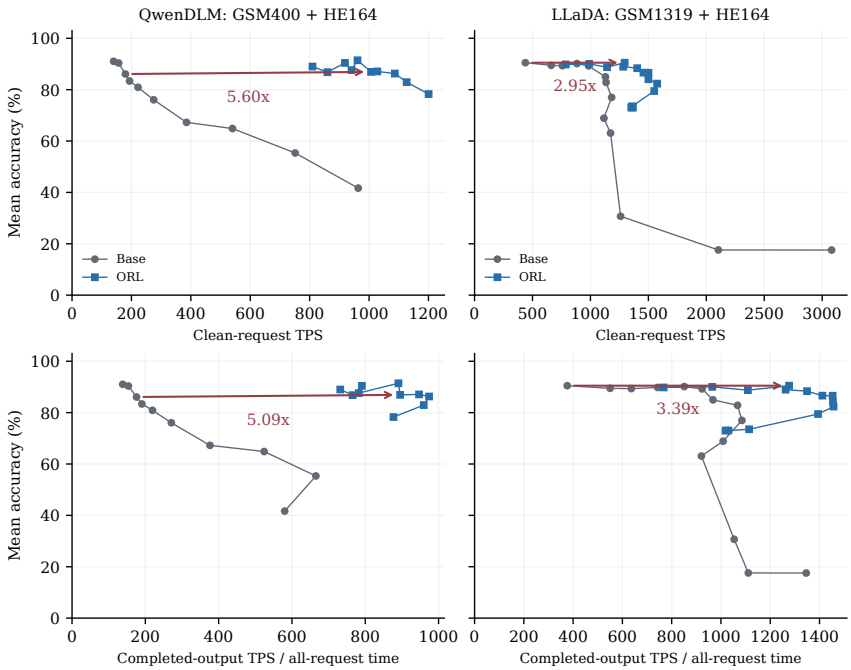


Figure 12: Common-task quality–throughput sweeps. Axes average GSM8K and HumanEval; all settings remain visible. Top: clean TPS; bottom: completed-output TPS charging failed-request time. The highlighted Qwen pair gives $86.09/86.92\%$ mean accuracy and $5.60/5.09\times$ clean/completed TPS gains. The highlighted LLaDA pair gives 90.49% on both sides, with $2.95/3.39\times$ gains. Lines follow the configuration order in the repository; matching refers to observed mean accuracy.

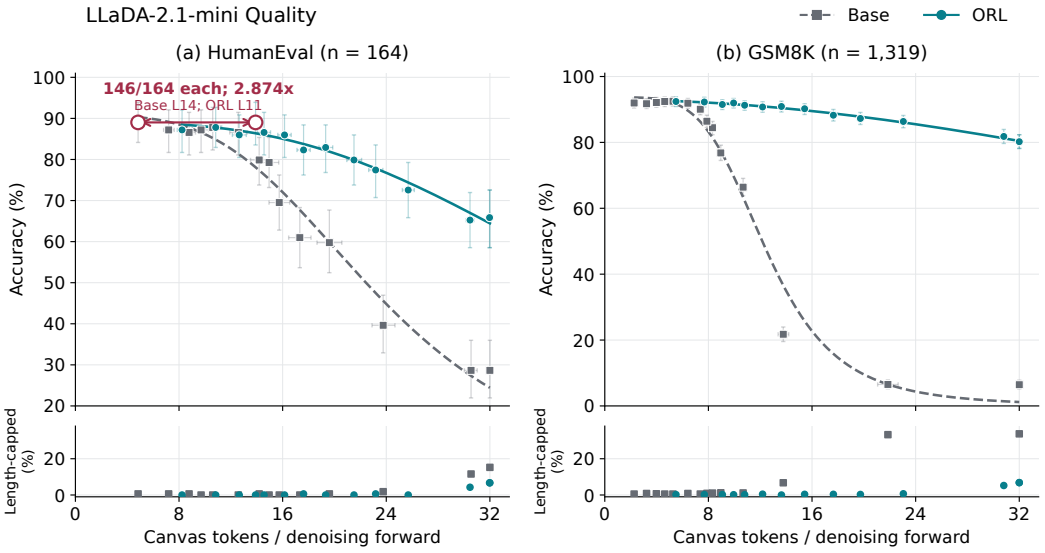


Figure 13: LLaDA Quality-mode Base versus ORL, with all operating points. Lines are independent log-logistic fits within observed ranges; bars are pointwise 95% percentile intervals from 2,000 paired task-vector resamples, conditional on the archived evaluation records. They quantify task-resampling uncertainty, excluding between-run variation and simultaneous coverage across operating points. The HumanEval arrow compares Base profile L14 with ORL profile L11; lower strips show length failures. Profile definitions are in the anonymous repository (Appendix E.1).

H Additional Ablations

H.1 System Cost

At $n = 16$ rollouts, packing reduces actor-update time from 368 to 131 s and replay materialization from 55 to 16 s; cost balancing further reduces the update to 96 s. Across rollout counts, update cost drops from 0.60–0.93 to 0.24–0.25 s per 1k supervised tokens. Packing makes each replay row larger but reduces row count by about $5\times$, so per-row cost alone is misleading. Asynchronous host staging does not consistently improve per-unit cost; the large rollout gain requires GPU-resident handoff.

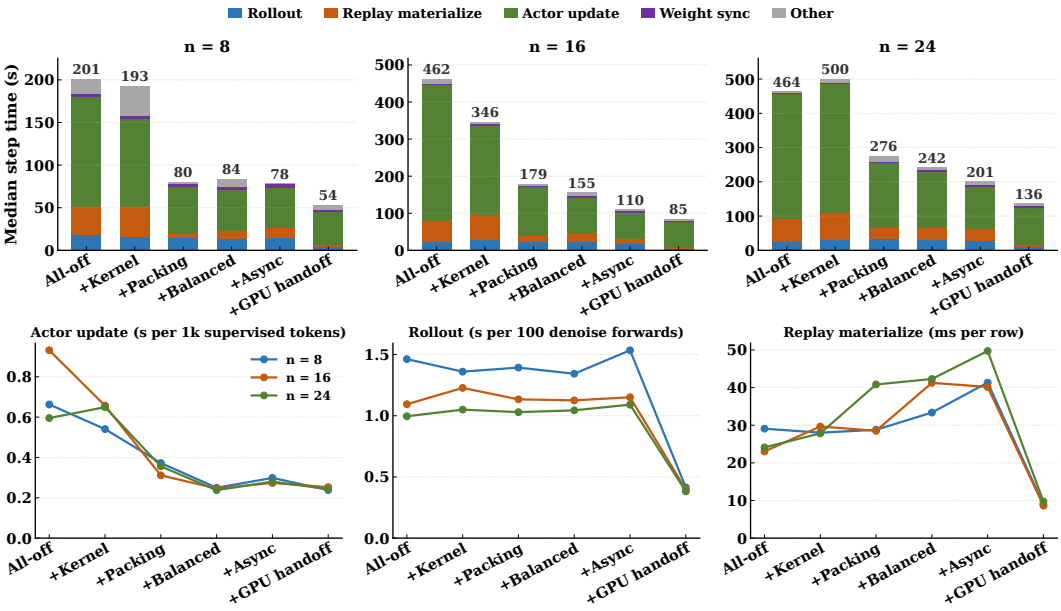


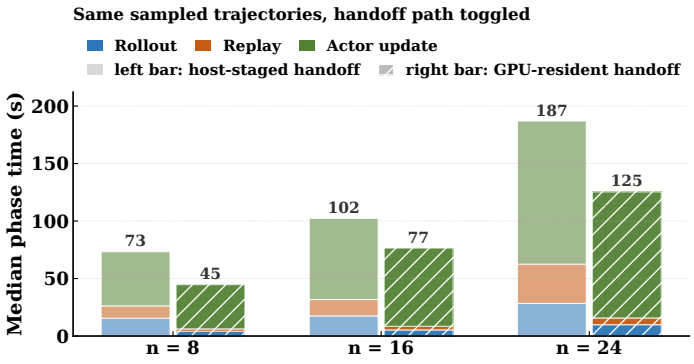
Figure 14: QwenDLM system ablation at response cap 2,048. Top: median phase times; bottom: median costs per 1k supervised tokens, 100 denoising forwards, and replay row. Medians use post-warmup steps; the main figure instead uses mean step times. Raw runs can differ in sampled workload.

Paired handoff and memory. On identical trajectories at $n = 8, 16, 24$, GPU-resident handoff changes rollout time from 15.5/17.4/28.4 to 4.2/5.3/9.9 s and replay materialization from 10.7/14.3/34.1 to 2.0/3.2/5.9 s, with little actor-update change. This isolates transport from workload variation. At cap 2,048, the full system uses about 19 GB GPU and 113–116 GB host memory; baseline host use grows to 381/541/799 GB. At cap 8,192, the baseline exceeds the node's 944 GB host capacity. The plotted 359 GB system point disables GPU handoff because the full variant reaches its capture-arena limit; it does not demonstrate constant host memory at that length.

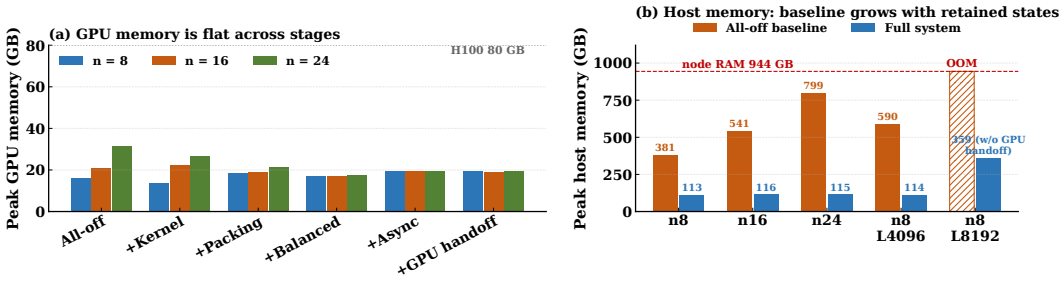
H.2 Residual Scaling and Block Size

Matched QwenDLM/GSM8K runs differ only in $\gamma = 8$ residual rescaling. Within their shared measurement window, excluding skipped updates, median $100\|\Delta W\|_2/\|W\|_2$ for the residual MLP is 0.00142% with scaling and 0.04431% without ($31.3\times$); backbone values are 0.00122%/0.00117% ($0.96\times$). These are measured optimizer-dependent update ratios.

The MBPP block-size study complements Figure 4: ORL raises TPF from 8.6 to 13.0 at block size 32 and from 7.2 to 16.6 at 128.

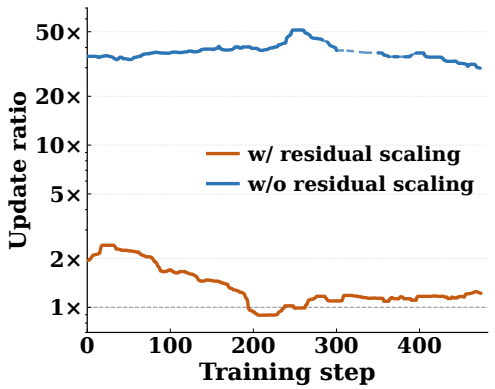


(a) Handoff paths on identical trajectories.

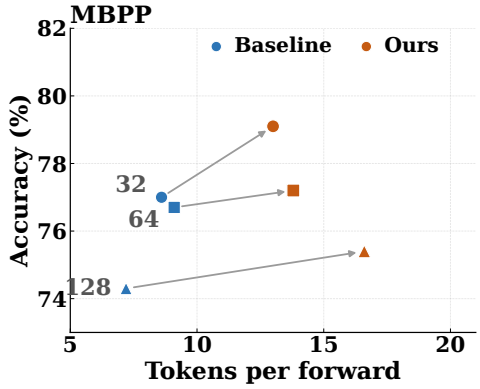


(b) GPU and host memory; 8,192-token caveat in text.

Figure 15: Transport and memory diagnostics. Paired handoff bars show median rollout/replay/update times; memory panels show peaks. All GPU configurations fit an 80 GB H100.



(a) Residual scaling controls relative updates.



(b) LLaDA training block size versus MBPP efficiency.

Figure 16: Residual-path stability and block-size sensitivity. Rescaling primarily changes residual, not backbone, update magnitudes.

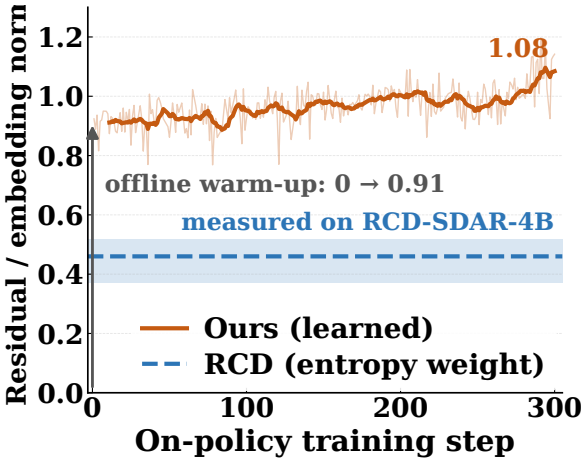


Figure 17: QwenDLM residual-to-embedding norm ratio, rising from 0.91 on the first on-policy rollouts to 1.08 during training. The dashed RCD-SDAR-4B reference is a separately measured entropy-weighted diagnostic. The offline-SFT RMS ratio of 1.06 at the same initializer uses a different population and statistic, so the values are not directly comparable. These traces describe residual use, not its causal benefit.

Table 7: AEC versus a naive entropy loss on QwenDLM: mean GSM8K accuracy (%) / TPF over independent training runs and repeated evaluations. This ablation trains only on math prompts because of compute limits.

Objective	GSM8K
AEC (λ_{AEC}=50)	**91.6** / 7.7
Naive entropy loss (λ=4)	90.0 / 7.1
Naive entropy loss (λ=50)	76.8 / **8.2**

H.3 AEC versus Naive Entropy Minimization

Table 7 compares AEC with a naive entropy loss that uniformly minimizes prediction entropy with weight λ. A small weight (λ=4) lowers both accuracy and TPF relative to AEC, while a large weight (λ=50) raises TPF only from 7.7 to 8.2 at a 14.8-point accuracy cost. Because AEC penalizes only steps whose unresolved entropy fails to contract, it reaches a better accuracy–TPF tradeoff. Because of compute limits, this ablation trains QwenDLM only on math prompts. Each configuration is trained independently multiple times and evaluated repeatedly. We average evaluations within each trained checkpoint, then average equally across training runs.

H.4 Staleness

The QwenDLM comparison in Figure 2a and Table 8 compares $N = 1, 25, 75$ using five independent training seeds for each N, with three repeated evaluations per trained checkpoint. Each evaluation uses 500 GSM8K tasks; training uses the 6,998-row subset. We first average accuracy and TPF over the three evaluations within each trained checkpoint, then average equally across the five training seeds. Variants share the initialization, verifier, and optimization objective. Each variant is trained until TPF stabilizes. Exact seeds and experiment scope are recorded in `configs/paper/staleness.yaml` in the artifact repository. N sets the periodic rollout-weight synchronization interval; every setting generates new trajectories, not a fixed offline cache. Supervised-token budgets are not matched: verified-response counts, retained states, and supervised-token totals depend on each variant's generated trajectories.

Table 8: Staleness results after TPF stabilizes: GSM8K accuracy (%) and TPF. QwenDLM entries average five independent training seeds, with three repeated evaluations per checkpoint; LLaDA-2.1-mini entries average independent training runs and repeated evaluations.

	QwenDLM		LLaDA-2.1-mini	
Setting	Acc. (%)	TPF	Acc. (%)	TPF
On-policy ($N = 1$)	89.8	13.7	92.0	8.2
$N = 25$	77.1	11.7	91.8	7.5
$N = 75$	78.1	9.9	90.2	5.8

H.5 Training Data and Compute

The data budget counts prompt and response tokens, including failed rollouts, without multiplying by training epochs. LLaDA training generates 5,000 responses: 0.74M prompt and 3.49M response tokens (4.2M total); 2.37M response tokens pass verification and enter the loss. Published budgets are about 40M/65M for d3LLM-LLaDA/Dream, 1B for Fast-dLLM-v2, and 54B for SDAR (Qian et al., 2026; Wu et al., 2026; Cheng et al., 2025). dParallel's 40M point is estimated from the same 92,428-trajectory data reused by d3LLM; counting only 256-token responses gives 23.7M (Chen et al., 2026c). We compare ORL's budget with post-training recipes, not AR-to-DLM conversion.

Table 9 counts every allocated H100 along the training chain leading to each reported checkpoint, including startup, model loading, rollout, verification, updates, and checkpoint writes. It excludes initializer production, checkpoint evaluation, other experimental runs, idle gaps, and attempts failing before their first checkpoint. Including the latter adds about 4.5 GPU-hours for QwenDLM and at most 8.5 for DiffusionGemma. When a job runs beyond the reported checkpoint, the next rollout timestamp bounds its endpoint to within one minute.

Table 9: Post-training compute for the reported checkpoints. Per-job hours follow chain order; GPU-hours count all allocated H100 80GB GPUs and are rounded.

Model	GPUs	Per job (h)	Total (h)	GPU-hours
QwenDLM	8	$5.00 + 0.66 + 1.79$	7.45	60
LLaDA-2.1-mini	8	$3.25 + 1.27$	4.52	36
DiffusionGemma	16	$8.22 + 9.55$	17.77	284

I Policy Dependence and Entropy

I.1 The Update and Stale Data

Let $w = (\theta, \phi)$ and let ν_w be the distribution of captured training records after verification, selection, and objective-specific weighting. At iteration u, ORL takes the frozen-data gradient

$$g_u = \nabla_w \mathbb{E}_{\xi \sim \nu_{w_u}} \ell(w; \xi)\big|_{w=w_u}. \tag{12}$$

Inputs, targets, and source entropies are held fixed for this differentiation. DiffusionGemma recomputes its source entropy during replay and then detaches it. Neither recipe differentiates through rollout generation or verification. In a differentiable common-density model, the full derivative of $\mathbb{E}_{\nu_w} \ell(w; \xi)$ would also contain $\mathbb{E}_{\nu_w}[\ell(w; \xi) \nabla_w \log \nu_w(\xi)]$. Thus verified CE is not itself a policy-gradient estimator of task reward.

Stale-gradient bound. At a fixed evaluation parameter w, assume $\|\nabla_w \ell(w; \xi)\|_2 \le G$ under both record distributions. Then

$$\|\mathbb{E}_{\nu_w} \nabla \ell(w; \xi) - \mathbb{E}_{\nu_v} \nabla \ell(w; \xi)\|_2 \le 2G \ \mathrm{TV}(\nu_w, \nu_v), \qquad \mathrm{TV}(P, Q) = \sup_A |P(A) - Q(A)|. \tag{13}$$

To prove this, project onto any unit vector and integrate the bounded scalar gradient against $\nu_w - \nu_v$, whose total variation norm is $2\,\mathrm{TV}(\nu_w, \nu_v)$. Taking the supremum proves the inequality. The

gradient bound is an assumption, not automatic for CE. The result bounds possible mismatch, not harm: different data distributions can give the same gradient. If $\nu_w \ll \nu_v$, importance weighting can correct the expectation in principle, subject to support and variance. Deterministic continuous residuals may instead have disjoint supports. Fresh collection is therefore a design choice, not a proof that offline learning is impossible.

Verification and skipped updates. With one prompt, G independent rollouts, pass probability $p_w(c)$, and equal verified-response weights, the expected CE/hybrid gradient per attempted batch is $[1-(1-p_w(c))^G]g_w^+(c)$, where $g_w^+(c)$ is the conditional verified gradient. Indeed, a nonempty verified batch has expected mean $g_w^+(c)$, and its probability is $1-(1-p_w(c))^G$. Skipping all-wrong batches therefore reweights prompts. This calculation does not directly apply to gated AEC's weighted ratio, dependent rollouts, or nonlinear optimizer updates.

I.2 Entropy and Token Acceptance

Use unnormalized entropy $h(p) = -\sum_v p_v \log p_v$; the method uses $H = h/\log|\mathcal{V}|$. Suppose proposals are greedy and acceptance uses the same distribution at fixed threshold $0 < \tau < 1$. For n eligible positions,

$$\frac{\#\{i : \max_v p_{iv} \le \tau\}}{n} \le \min\left\{1, \frac{\bar{h}}{-\log\tau}\right\}. \tag{14}$$

Each failing position contributes at least $-\log\tau$ because $h(p_i) \ge -\log\max_v p_{iv}$; summing proves the bound. This concerns confidence, not correctness, and does not directly cover different confidence temperatures, sampled proposals, entropy budgets, or re-noising decoders.

If actual rollout means satisfy $H_{k+1} \le rH_k + e_k$ with $0 < r < 1$, induction gives $H_k \le r^k H_0 + \sum_{j<k} r^{k-1-j} e_j$. If all unresolved positions are eligible, all above-threshold greedy proposals commit, and positions never reopen, the remaining count obeys

$$n_K \le n_0 \prod_{k<K} \min\left\{1, \frac{H_k \log|\mathcal{V}|}{-\log\tau}\right\}. \tag{15}$$

The vocabulary factor converts normalized entropy to h. A right-hand side below one implies completion, not a correct answer. A finite AEC penalty on selected training pairs does not impose the assumed contraction on new rollouts.

Changing masks also changes the measurement. If $U' = U \setminus A$ with both A, U' nonempty, then

$$\bar{h}_{k+1}(U') - \bar{h}_k(U) = [\bar{h}_{k+1}(U') - \bar{h}_k(U')] + \frac{|A|}{|U|}[\bar{h}_k(U') - \bar{h}_k(A)]. \tag{16}$$

Removing confident positions can raise the remaining mean without worsening any prediction. Conversely, temperature sharpening lowers entropy: $dh(\mathrm{softmax}(\beta z))/d\beta = -\beta \operatorname{Var}_{p_\beta}(z) \le 0$, while leaving a wrong argmax wrong. These are reasons to measure quality separately.

I.3 Optimizer and Rescaling

At a fixed captured input and mask, $p = \mathrm{softmax}(z)$ gives

$$\Delta h = -\operatorname{Cov}_{v\sim p}(\log p_v, \Delta z_v) + O(\|\Delta z\|_2^2). \tag{17}$$

This follows from $\partial h/\partial z_v = -p_v(\log p_v + h)$. The actual Δz depends on the network Jacobian, all losses, clipping, and optimizer state; a policy-gradient covariance specialization need not describe CE/AEC/AdamW. Rollout-average changes additionally include state-distribution shift.

For a compensated projection $\widetilde{W} = \gamma W$ evaluated as $\widetilde{W}/\gamma$, the chain rule gives $\nabla_{\widetilde{W}} L = \nabla_W L/\gamma$. Equal-rate SGD reduces the effective W update by γ^2. Adam with consistently scaled moments and negligible epsilon reduces it by approximately γ, not γ^2; clipping, untransformed moments, and epsilon can change this relation. AdamW's relative decay is unchanged. Equation 6 proves forward-function preservation; stability and the measured update ratios remain empirical.

J ACCURACY–TPF CURVES

J.1 A DESCRIPTIVE MODEL

For an operating point τ, write $S = \mathbf{1}\{\text{not length-terminated}\}$, Z for benchmark correctness, and $U = C/D$. The sweep estimates $a(\tau) = \mathbb{E}[SZ]$ and $t(\tau) = \mathbb{E}[U \mid S = 1]$, not $P(Z = 1 \mid U = t)$. In particular, $a = (1 - P(S = 0))P(Z = 1 \mid S = 1)$; collapse changes both axes. For the canvas diagnostic, replace C by BK throughout.

We use the log-logistic family

$$a(t) = \frac{A}{1 + (t/f_0)^\kappa}, \qquad 0 < A \le 1, \quad f_0, \kappa > 0. \tag{18}$$

One sufficient latent model assumes a solvable fraction A, sensitivity $\Lambda \sim \mathrm{Exp}(\eta)$ within that fraction, and retention $\exp(-\Lambda t^\kappa)$. Then

$$A \int_0^\infty e^{-\lambda t^\kappa} \eta e^{-\eta\lambda}\, d\lambda = \frac{A\eta}{\eta + t^\kappa}, \qquad f_0 = \eta^{1/\kappa}.$$

Exponential sensitivity and power exposure are assumptions, not consequences of residual decoding. A gamma sensitivity instead gives $A(1 + t^\kappa/\eta)^{-r}$. Neither identity explains why an aggregate, termination-conditioned TPF should equal the assumed exposure.

The curve has plateau A, half-plateau location f_0, and $da/d\log t|_{t=f_0} = -A\kappa/4$. At relative retention q,

$$t_q = f_0 \left(\frac{1-q}{q}\right)^{1/\kappa}, \qquad \frac{t_i(q)}{t_j(q)} = \frac{f_i}{f_j}\left(\frac{1-q}{q}\right)^{1/\kappa_i - 1/\kappa_j}. \tag{19}$$

Thus a quality-independent horizontal factor requires shared κ. At common absolute accuracy α, $t_m(\alpha) = f_m(A_m/\alpha - 1)^{1/\kappa_m}$. The ratio is constant on an open accuracy interval only if both plateaus and shapes match: differentiating its logarithm gives

$$\frac{d\log(t_R/t_B)}{d\alpha} = \frac{A_B}{\kappa_B\alpha(A_B - \alpha)} - \frac{A_R}{\kappa_R\alpha(A_R - \alpha)};$$

equating polynomial coefficients after cross-multiplication requires $\kappa_B = \kappa_R$ and $A_B = A_R$. Otherwise speed ratios must name their accuracy level. When each block requires a denoising forward, canvas TPF is at most B; $f_0 > B$ is an extrapolated parameter, not an attainable operating point. Even $t_{0.9}$ requires a measured-range check.

J.2 LOCKED LLADA SWEEP

All 56 Base/ORL points are reconstructed from 41,524 request records: fourteen Quality-mode profiles on HE164 and GSM1319 for each checkpoint. Task IDs match across profiles/checkpoints; accuracy, clean counts, and both TPF definitions reproduce saved aggregates within 10^{-10}. HumanEval uses base tests and GSM8K its original extractor.

Observed operating points. The HumanEval pair in Figure 13 has a $2.874\times$ canvas-TPF ratio at 146/164 correct on both sides. Recomputing on the 163 tasks clean for both gives $2.883\times$. For 20,000 paired task bootstraps, the canvas-ratio interval is $[2.737, 3.012]$ and the accuracy-difference interval is $[-4.268, 4.268]$ percentage points. These pointwise intervals condition on the archived evaluation records and the displayed pair; they quantify task-resampling uncertainty, without estimating between-run variability or accounting for comparisons across the full sweep. They do not establish population equivalence. On GSM8K, maximizing observed canvas TPF subject to empirical accuracy floors of 90%/92% gives ratios 2.093/1.461. The resulting operating points need not have identical accuracies.

Shared evaluation profiles. Table 10 compares Base and ORL under identical settings and retains every operating point, including quality losses. Profile definitions are provided in the anonymous repository (Appendix E.1). This comparison does not isolate AEC from the other training components.

Table 10: LLaDA Base–ORL comparisons across all fourteen shared profiles. Δa: ORL minus Base accuracy in percentage points. R: clean *completion*-TPF ratio, not padded canvas or wall TPS.

	HumanEval		GSM8K	
Profile	Δa	R	Δa	R
L01	+37.20	1.048	+73.77	0.920
L02	+37.20	1.096	+73.69	1.380
L03	+25.61	1.344	+60.05	2.190
L04	+12.80	1.368	+19.94	2.111
L05	+16.46	1.392	+10.46	2.159
L06	+10.37	1.423	+3.79	2.077
L07	+3.66	1.352	+3.79	1.914
L08	+2.44	1.296	+0.83	1.812
L09	-0.61	1.346	-1.14	1.862
L10	-1.22	1.432	-1.21	2.011
L11	+1.83	1.500	-0.45	2.110
L12	-0.61	1.508	-0.61	2.235
L13	+0.61	1.574	+0.53	2.334
L14	-1.83	1.774	+0.45	2.379

At profile L11, HE accuracy is 143/164 versus 146/164 and completion TPF is 8.720 versus 13.077; GSM accuracy is 1219/1319 versus 1213/1319, with TPF 4.045 versus 8.535. Across 20,000 joint task-vector bootstraps (seed 20260925), a max-standardized-deviation band over 56 accuracy/log-ratio coordinates gives TPF-ratio intervals [1.385,1.623]/[2.059,2.163] and accuracy-difference intervals [-6.00,9.66]/[-2.74,1.83] for HE/GSM. These approximate familywise intervals cover the displayed profiles and benchmarks conditional on the evaluated checkpoints and archived request records. They quantify task-resampling uncertainty without estimating variation across independent training runs or repeated evaluations.

Held-out operating points. We fit each checkpoint/benchmark independently with a marginal binomial working loss, four initializations, and bounds $A \in [.001, 1]$, $f_0 \in [.1, 500]$, $\kappa \in [.1, 50]$. Each fold excludes one operating point or one adjacent pair in the ordered grid, predicting all fourteen points once. Both predictors receive held-out measured canvas TPF. The comparator is decreasing isotonic regression with linear interpolation, endpoint clamping, and pooled weights at duplicate coordinates.

Table 11: Conditional accuracy-prediction RMSE (percentage points), including endpoint folds. This holds out operating points, not tasks or seeds.

	One point out		Adjacent pair out	
	Log-logistic	Isotonic	Log-logistic	Isotonic
HE / Base	4.46	3.66	6.65	5.76
HE / ORL	1.69	1.42	2.39	1.31
GSM / Base	6.91	8.28	7.44	9.66
GSM / ORL	0.46	0.44	0.59	0.89

The log-logistic fit is not uniformly better. In 400 paired task-vector bootstrap refits, 95% intervals for isotonic minus log-logistic adjacent-pair RMSE are [-4.18,3.00]/[-2.04,0.22] for HE Base/ORL and [1.10,3.30]/[-0.06,0.37] for GSM Base/ORL. Only GSM Base's individual interval excludes zero in favor of the curve; these intervals are exploratory and not simultaneous. Full-data Base/ORL shape estimates are 3.39/3.02 on HE and 4.64/2.02 on GSM, not evidence of a shared shape. ORL's half-plateau estimates 44.03/80.39 exceed the canvas bound 32. The curve is a compact empirical approximation, not an identified latent-difficulty law or proof of an AEC mechanism.