# ZENO: ZONE EXPANDING DECODING OPERATOR FOR DIFFUSION LANGUAGE MODELS

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

## ABSTRACT

Diffusion Large Language Models (dLLMs) generate text by unmasking tokens from anywhere on the generation canvas. This maximizes generation flexibility but minimizes context buildup by forcing models to commit to premature conclusions before establishing reasoning chains. This lack of directional grounding leads to poorly formed trajectories, high inference latency and rigid denoising schedules. We introduce **ZENO** (**Z**one **E**xpanding decodi**N**g **O**perator), a training-free sampler that transitions from autoregressive grounding to parallel diffusion within a single generation instance at inference time. ZENO enforces early directional priors by restricting generation to a geometrically expanding prefix window, then shifts to a fixed block once sufficient context crystallizes. Evaluated on six standard benchmarks, across base and instruction-tuned checkpoints of LLaDA and Dream families, ZENO achieves up to a $4.3\times$ and $4.0\times$ speedup in per-query latency, with token throughput gains of up to $6.2\times$ and $20.2\times$ and cuts reverse diffusion steps by up to $78\%$. ZENO also improves generation quality, yielding a mean accuracy gain of up to $12.8$ points over vanilla decoding. Furthermore, enabling KV caching reduces per-query latency by up to $7.9\times$ with small accuracy trade-offs, establishing ZENO as a fast and efficient decoding strategy for dLLMs.

## 1 INTRODUCTION

Large Language Models (LLMs) have achieved remarkable capabilities across language understanding, reasoning and generation tasks (Brown et al. 2020; Zhao et al. 2026). Currently, the dominant paradigm remains autoregressive (AR) decoding. However, because AR models generate text strictly in left-to-right order one token at a time, their inference throughput is fundamentally bottlenecked by this sequential nature. But recently, diffusion Large Language Models (dLLMs) have emerged as a compelling alternative (Austin et al. 2023a; Ho et al. 2020b). By replacing AR models' sequential left-to-right generation with iterative bidirectional refinement, dLLMs enable parallel token unmasking and global context conditioning at every step (Nie et al. 2025; Sahoo et al. 2024). However, this paradigm shift from AR to diffusion comes with a structural cost. Diffusion decoding replaces the single sequential pass of an AR model with a long reverse diffusion chain, where the step budget and transfer schedule must be set a priori.

While a growing body of work on training-free test-time scaling methods tries to reduce the cost, aiming to speed up generation by dynamically exiting or altering the schedule, these methods generally force a strict trade-off, sacrificing generation quality for speed. More critically, unconstrained diffusion generation suffers from a severe lack of a directional prior, resulting in poor reasoning and systematic errors in the generation trajectory. Standard diffusion decoding starts from a fully masked sequence and iteratively refines it toward a target by unmasking tokens across the entire canvas, with no notion of guided directionality. In the high-dimensional search space of complex reasoning, the absence of a strong prior often diverges the generative trajectory. This acts much like an unbiased random walk in a high-dimensional space (A very popular idea in statistical physics and probability theory), which has a vanishingly low probability of reaching a specific destination (simulations provided in Appendix A). Furthermore, as noted in recent analysis of diffusion generation dynamics (e.g., Prophet (Li et al., 2026)), diffusion language models frequently "generate the answer beforehand," committing to a premature conclusion somewhere on the canvas before the intermediate reasoning steps are fully formed. These observations reveal that unconstrained global denoising suffers from structural blind spots. In this work, rather than treating AR and diffusion as



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.

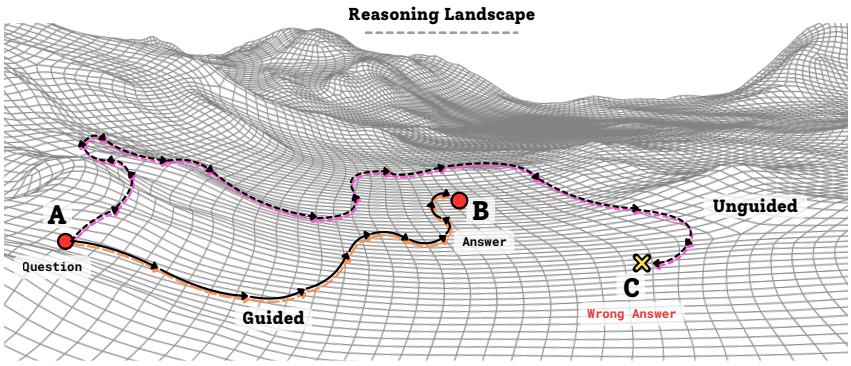



Figure 1: Guided generation with limited degree of freedom in initial stages vs unguided generation.

two competing strategies, we ask whether decoding needs to commit to a single paradigm at all. We hypothesize that for diffusion models, guiding early generation steps by constraining the degrees of generative freedom establishes a robust logical foundation, acting as a strong prior (e.g. biased random walk; Figure 6; Appendix A). By slowly increasing the diffusion generation window, we allow the model more degrees of freedom as the generation progresses. This restricts the effective search space early on in the generation process and prevents premature hallucinations that lead to incorrect responses. Consequently, this allows the total number of generation steps to significantly reduce later in the generation process by confidently unmasking multiple correct tokens, improving generation speed and accuracy simultaneously.

Harnessing these dynamics, we introduce **ZENO** (**Z**one-**E**xpanding Decodi**N**g **O**perator) for Diffusion Language Models, an inference-time decoding strategy that binds autoregressive grounding with bidirectional diffusion refinement within a single, dynamically evolving generation instance. We evaluate ZENO across base and instruction-tuned diffusion models, spanning six benchmark suites including reasoning tasks, coding and general multi-choice question answering. The reported results show substantial decoding savings and strong accuracy gains on several instruction-tuned reasoning tasks, with more variable accuracy effects on base checkpoints and knowledge benchmarks. Our main contributions can be summarized as:

**1. A dynamic fusion of AR–to-diffusion decoding paradigm switch.** We introduce **ZENO**, a functional decoder that performs inference-time, on-the-fly paradigm switching between autoregressive and diffusion decoding, guided by a geometrically expanding view of the local sequence, requiring no additional training or auxiliary models.

**2. Simultaneous efficiency and quality gains.** Unlike prior training-free inference-time accelerators that trade quality for speed, ZENO improves both. Our primary method, ZENO(NB)+C improves mean accuracy by $11.3 - 11.4$ points on Instruct models and around $+3.7\times$ average per-query latency speedup compared to the vanilla baseline. Furthermore, paired with KV caching, this boost reaches up to $+7.9\times$ average per-query speedup with minimal accuracy loss.

**3. Broad empirical validation across the board.** Most prior work reports improvements only in hand-picked few-shot settings on either Base or Instruct variant. We propose ZENO as a model-class invariant scheduler and evaluate on a broad spectrum of tasks spanning multiple subject areas, showing consistent and substantial improvements in both Base and Instruct variants.

## 2 Background and Related Work

**Discrete Diffusion Models.** Diffusion models, initially popularized in continuous domains such as image generation (Sohl-Dickstein et al., 2015; Ho et al., 2020a; Song et al., 2021), have recently emerged as a compelling alternative to standard AR models for language generation. Early adaptations of discrete state-space models laid the theoretical groundwork by utilizing continuous-time Markov chains and discrete score matching (Austin et al. 2023b; Hoogeboom et al. 2022; Camp-

bell et al. 2022; Lou et al. 2024). Over time, this groundwork led to the development of Masked Diffusion Models (MDLMs) (Sahoo et al. 2024; Shi et al. 2025), which generate text by leveraging bidirectional context to iteratively predict and unmask tokens from a fully masked sequence. Recent large-scale dLLMs, such as LLaDA (Nie et al. 2025) and Dream (Ye et al. 2025) have demonstrated their capability to match the performance of similarly sized AR models (e.g., Llama, Mistral, etc.). Moreover, their bidirectional nature inherently circumvents AR model limitations, such as the reversal curse (Berglund et al. 2024). Despite their advantages, a central bottleneck for MDLMs remains in their high inference latency, driven by the need for iterative multi-step denoising across the entire sequence length. Therefore, addressing this efficiency gap has become a primary focus area for deploying discrete diffusion models in real-world, latency-critical applications.

**Test-time Sampling Strategies for dLLMs.** The vanilla MDLM sampler unmasks a fixed number of tokens per step across the whole canvas, which increases the generation cost and latency. To mitigate the high inference cost of MDLMs, a significant body of current work has focused on optimizing the decoding and sampling process without architectural changes or additional fine-tuning. Recent literature tries to address this across three primary axes:
**(1) Adaptive Token Commitment:** Rather than unmasking a fixed number of tokens per step, these methods dynamically adjust how many positions to commit per step. Fast-dLLM (Wu et al. 2025) commits tokens in parallel based on confidence thresholds. SlowFast sampling (Wei et al. 2026) divides decoding into exploratory and accelerated phases, forcing commitments based on certainty and convergence. Recognizing confidence alone to be brittle, KLASS (Kim et al. 2026) gates token unmasking using a joint threshold of predictive confidence and KL-divergence stability across steps.
**(2) Sequence Termination (Early Exit):** This line of work halts diffusion entirely once the sequence has converged. Prophet (Li et al. 2026) introduces a training-free early commit rule based on the aggregate top-2 logit gap, freezing a sequence once a strict threshold is met. SchED (Mohamed et al. 2025), on the other hand, smooths this trigger into a dynamic decay schedule that relaxes confidence thresholds over time.
**(3) Block-wise Decoding:** To recover the efficiency of Key-Value (KV) cache lost in full sequence diffusion, Block diffusion sampler (Arriola et al. 2025) segments the output canvas into blocks. They execute parallel diffusion within blocks while stepping autoregressively between them and moving forward, drastically speeding up inference.

# 3 METHODOLOGY

## 3.1 THE PRIOR ART

**Masked diffusion decoding.** Let $\mathcal{V}$ be the token vocabulary and $\texttt{[MASK]} \notin \mathcal{V}$ be a unique mask token. A masked diffusion language model operates on a canvas $x \in (\mathcal{V} \cup \{\texttt{[MASK]}\})^T$ of length $T$, with $x_i$ being the token at position $i$, comprising a prompt of length $P_r$ followed by a generation region $\mathcal{G} = \{p, \ldots, T-1\}$ of length $L = T - P_r$. We define $x^{(t)}$ for the canvas after $t$ of the $S$ nominal denoising steps, so that $x_i^{(t)} \in \mathcal{V} \cup \{\texttt{[MASK]}\}$ is the token occupying position $i$ at that stage. The decoding starts from a fully masked generation region, $x_i^{(0)} = \texttt{[MASK]}$ for all $i \in \mathcal{G}$, and $\mathcal{M}^{(t)} = \{\, i \in \mathcal{G} : x_i^{(t)} = \texttt{[MASK]} \,\}$, which denotes the positions still masked after $t$ steps, so that $\mathcal{M}^{(0)} = \mathcal{G}$. At timestamp $t$, the model unmasks a subset of the remaining masked positions anywhere in $\mathcal{G}$ and once unmasked, a position is frozen.

**Candidates, confidence, and selection.** A network $f_\theta$ maps the canvas to per-position logits in one forward pass, and a candidate token is drawn at every position by Gumbel-max sampling,

$$\hat{x}_i^{(t)} = \arg\max_{v \in \mathcal{V}} \left(\ell_{i,v}^{(t)} + \tau_{\text{gen}}\, g_{i,v}\right), \qquad g_{i,v} \sim \text{Gumbel}(0,1), \tag{1}$$

These are optionally sharpened with classifier-free guidance (Ho & Salimans 2022). Candidates are proposals only. They are recomputed every step until unmasked, and each carries a confidence score under low-confidence remasking.

$$c_i^{(t)} = \text{softmax}\big(\ell_i^{(t)}\big)_{\hat{x}_i^{(t)}} \in [0,1], \tag{2}$$

Random remasking decouples this choice entirely from the model by sampling from $c_i^{(t)} \sim \mathcal{U}(0,1)$.

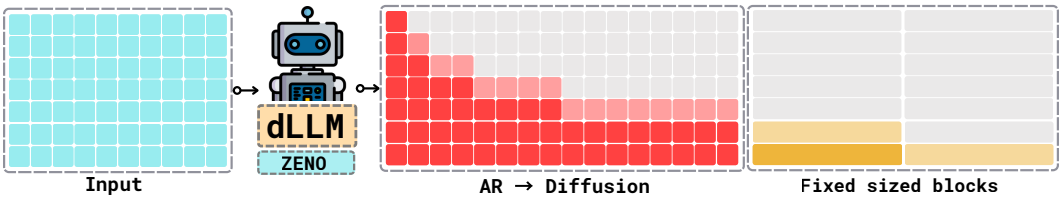



Figure 2: **ZENO's** sampling process. The AR → Diffusion section slowly opens up the canvas to decode in diffusion, starting from a narrow window; the latter part denoises with fixed block sizes.

Decoding runs in phases of $S'$ steps sharing a fixed eligibility set $\mathcal{E}^{(t)} \subseteq \mathcal{M}^{(t)}$; each step unmasks and freezes the top-scoring share of that phase's eligible positions, evenly spread so nothing eligible is left masked once the phase ends. A sampler is fully specified by $\mathcal{E}^{(t)}$ and this selection rule: the vanilla MDLM sampler sets $\mathcal{E}^{(t)}$ to a fixed-size block of $\mathcal{G}$, filled in order by confidence. A collection of all parameter definitions is provided in Appendix K.

### 3.2 **ZENO's** Sampling Process

Diffusion decoding hands the model the entire canvas from the first step, so its freedom is widest exactly when its prior is weakest. Fixed blocks address this with a constant width, but that never adapts to how much has already been settled. ZENO instead controls freedom to accumulate a strong prior by using a single zone anchored at the start of the generation region, a few tokens wide at first and widened once it is frozen or its pass budget is spent. This fusion of AR to Diffusion switching grounds the trajectory while it is still cheap to steer.

**Zone schedule.** Three parameters govern how the zone opens up for ZENO. The initial width $w_0 \geq 1$ sets how much freedom the model is given at the very first step; the growth base $\gamma > 1$ sets how fast that freedom expands once a zone is filled, and the skip factor $\kappa \geq 0$ thins the schedule by retaining only every $(\kappa+1)$-th power of $\gamma$, trading a finer ramp for fewer phases. Combining these three parameters provides a broad control over the scheduling dynamics. We define the geometric schedule $\mathcal{W}^{\mathrm{N}} = (w_1 < w_2 < \cdots < w_K)$ of cumulative zone widths as

$$w_m = \max\Big(1,\ \big\lfloor w_0\, \gamma^{\,(m-1)(\kappa+1)} \big\rceil\Big), \qquad m = 1, 2, \ldots, \tag{3}$$

where $\lfloor \cdot \rceil$ rounds to the nearest integer, keeping strictly increasing values and truncating at $L$ (the maximum generation window). The schedule starts at $w_1 = w_0$ and ends at $w_K \equiv L$. Phase $m$ runs with the zone $\mathcal{Z}$,

$$\mathcal{Z}_m = \{\, i \in \mathcal{G} \ :\ i - p < w_m \,\}, \qquad \mathcal{Z}_1 \subset \mathcal{Z}_2 \subset \cdots \subset \mathcal{Z}_K = \mathcal{G}, \tag{4}$$

so the zones are nested, every phase inherits the positions its predecessors settled, and phase $m$ newly exposes $\Delta_m = w_m - w_{m-1}$ positions, with $\Delta_1 = w_1$.

Geometric growth is the right behavior early, when the prefix is short, but its increment scales with the zone, so the final phase of Eq. 3 exposes $L(1 - 1/\gamma)$ positions at once and the schedule ends in the wide, weakly grounded regime it was built to avoid. We therefore cap the increment once the frozen prefix is long enough to constrain what follows: given a switch width $W^\star$ and a constant post switch block length $\beta$, the hybrid schedule $\mathcal{W}^{\mathrm{NB}}$ follows Eq. 3 up to $W^\star$ and advances in fixed steps thereafter. The composite geometric scheduler operator can therefore be defined as,

$$w_{m+1} = \begin{cases} \min\big(L,\ \big\lfloor w_m\, \gamma^{\kappa+1} \big\rceil\big), & w_m < W^\star \quad \text{(geometric)}, \\ \min\big(L,\ w_m + \beta\big), & w_m \geq W^\star \quad \text{(constant)}. \end{cases} \tag{5}$$

Following this scheduler, no phase past the switch is asked to ground more than $\beta$ new positions at a time, while the cheap ramp over the answer prefix is left intact. Regardless of the schedule, its phase is partitioned evenly into $K$ equal split, each comprising $S' = \lceil S/K \rceil$ steps each. Each phase spreads the $\Delta_m$ positions exposed by its new zone across $S'$ steps, unmasking $k_t$ positions at step $t$ with the constraint that $\sum_{t=1}^{S'} k_t = \Delta_m$."

**Unmasking rule.** While phase $m$ runs, a position is eligible only if it is both masked and inside the current zone ($\mathcal{E}^{(t)} = \mathcal{M}^{(t)} \cap \mathcal{Z}_m$). Eligibility is enforced on the scores of Eq. 2 before anything is selected: $\tilde{c}_i^{(t)} = c_i^{(t)}$ for $i \in \mathcal{E}^{(t)}$ and $\tilde{c}_i^{(t)} = -\infty$ otherwise. This condition, caps positions to be pulled into the selected set until the zone reaches it. The selected set $R_t$ then follows one of two rules. Schedule-driven selection unmasks exactly $k_t$ eligible positions with the highest-scoring eligible scores,

$$R_t = \arg \max_{R \subseteq \mathcal{E}^{(t)},\, |R| = k_t} \sum_{i \in R} \tilde{c}_i^{(t)}, \tag{6}$$

Since the per-step counts $k_t$ sum to $\Delta_m$, every zone is completely unmasked by its final allotted step. Under this mode, a subsequent phase never begins with an incomplete prefix, ensuring conditioning rests on fully resolved prefixes. Confidence-threshold selection instead unmasks all eligible positions whose model confidence meets or exceeds a predefined threshold $\tau \in [0, 1]$,

$$R_t = \left\{ i \in \mathcal{E}^{(t)} \;:\; c_i^{(t)} \geq \tau \right\}, \tag{7}$$

Under this condition, a single pass has the freedom to unmask any number of positions rather than the fixed $k_t$, focusing computational passes where the sequence remains uncertain. To ensure stability, this is paired with a per-step cap $M$ and a forced-progress fallback mechanism enabling a phase to terminate early once the active zone is clear. We evaluate on three proposed scheduler configurations: **`ZENO(N)`** : which grows the window by a fixed multiplicative factor and slowly increases the degree of freedom; **`ZENO(NB)`** : which has a cutoff after which it grows in fixed-sized blocks; and **`ZENO(NB)+C`**: which raises the confidence-based commitment cap (Appendix H). Comprehensive experimental evaluation and analysis are provided in the following sections.

## 4 Experimental Settings

**Baselines and Evaluation Metrics.** For comparison, we use four models from two different model families: LLaDA-8B-Instruct, LLaDA-8B-Base (Nie et al. 2025) and Dream-7B-Instruct, Dream-7B-Base (Ye et al. 2025). We benchmark three variants of ZENO against multiple methods, including (i) Vanilla diffusion (Sahoo et al. 2024) (full canvas generation baseline), (ii) Block (Arriola et al. 2025) (fixed block-causal linear scheduler baseline), (iii) Prophet (Li et al. 2026), (iv) SlowFast (Wei et al. 2026), (v) KLASS (Kim et al. 2026) and (vi) Sched (Mohamed et al. 2025). For datasets, we use GSM8K (Cobbe et al. 2021), SVAMP (Patel et al. 2021) for mathematics; MMLU (Hendrycks et al. 2021a) and MMLU-Pro (Wang et al. 2024) for general knowledge and reasoning. Finally, for code evaluation, we use MBPP (Austin et al. 2021) and HumanEval (Chen et al. 2021). Together, they provide a diverse evaluation set to validate the effectiveness of our method. For evaluation, we use accuracy as our primary metric. In addition, we measure tokens/s, speed/question, and NFE (Number of Function Evaluations) as additional validation metrics. Refer to Appendix C for metric details and Appendix D for dataset construction and prompting.

**Implementation Details.** All our experiments are run on a single node of 8 $\times$ NVIDIA A100 GPUs with PyTorch as the standard coding bench and the dLLM (Zhou et al. 2026) evaluation setup. We keep the maximum token generation limit ($L$) at 256 and the maximum number of diffusion steps or step budget ($S$) at 256 for all the models as the default for fair comparison, unless specified otherwise. For the comparison baselines, we use the best parameter settings provided in the respective papers. The seed is set to 42 and the decoding temperature to zero for deterministic output generation. None of the hyperparameters are tuned for specific datasets. Implementation details and the full hyperparameter configuration are provided in Appendix E and Table 16.

## 5 Results

### 5.1 Main Results

Table 1 reports accuracy and function evaluations (NFE) across all evaluated models and benchmarks. ZENO offers competitive accuracy with substantially fewer decoding steps, with the largest accuracy gains over vanilla decoding observed on instruction-tuned models. Among its three variants, ZENO(NB) and ZENO(NB)+C provide the strongest overall accuracy–efficiency trade-offs:

the former generally favors accuracy, while the latter further reduces NFE through confidence-based parallel unmasking. On base models, accuracy gains are smaller and task-dependent, while decoding remains substantially more efficient than vanilla sampling.

The strongest improvements occur on mathematical reasoning tasks. On LLaDA-Instruct, ZENO(NB)+C achieves the highest GSM8K accuracy among the evaluated methods at 75.3%, improving on Vanilla by 20.5 percentage points while reducing NFE from 256.0 to 56.9. On Dream-Instruct, ZENO(NB) reaches 79.8% on GSM8K and 87.7% on SVAMP, using 110.4 and 102.7 NFE, respectively—less than half the forward-pass count of Vanilla. ZENO(NB)+C further reduces these counts to 63.0 and 49.5 NFE, with accuracies of 76.3% and 86.3%, respectively.

These gains are not uniform across task types. On MMLU and MMLU-Pro, the primary ZENO variants sometimes trade accuracy for lower NFE. For example, on Dream-Instruct, ZENO(NB) trails Vanilla by 5.1 and 2.4 percentage points on these benchmarks, respectively, while reducing NFE from 256.0 to 136.7 on both. Overall, the results demonstrate substantial decoding savings, accompanied by strong accuracy improvements on instruction-tuned mathematical reasoning tasks and more variable outcomes on knowledge and code benchmarks.

For code generation tasks across MBPP and HumanEval, ZENO closely matches or exceeds all competitive baselines. Notably, on Dream-Instruct, ZENO(NB) delivers 54.3% on HumanEval (+25.0 points over Vanilla), while ZENO(NB)+C achieves a competitive 48.8% in only **29.3 NFE** (more than 8.7× step reduction). These empirical gains highlight that ZENO's dynamic scheduling acts as an implicit regularizer for steering parallel generation toward logically cohesive reasoning directions that iterative masking heuristics often break. Detailed output analysis for all the datasets is provided in Appendix I.

| Model | Method | Math | | | | General Knowledge | | | | Coding | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GSM8K | | SVAMP | | MMLU | | MMLU-Pro | | MBPP | | HumanEval | |
| | | Acc%↑ | NFE↓ | Acc%↑ | NFE↓ | Acc%↑ | NFE↓ | Acc%↑ | NFE↓ | Acc%↑ | NFE↓ | Acc%↑ | NFE↓ |
| LLaDA Instruct | Vanilla | 54.8 | 256.0 | 45.0 | 256.0 | 57.0 | 256.0 | **38.3**$^{tie}$ | 256.0 | 30.5 | 256.0 | 26.2 | 256.0 |
| | Block | 27.5 | 256.0 | 57.3 | 256.0 | 51.5 | 256.0 | 27.9 | 256.0 | 28.5 | 256.0 | 17.7 | 256.0 |
| | SlowFast | 71.5 | 123.1 | 84.3$^{tie}$ | 126.6 | **57.2** | 150.2 | **38.3**$^{tie}$ | 154.2 | 36.2 | 110.9 | **34.8** | 70.5 |
| | KLASS | 72.0 | 132.1 | 84.0 | 125.4 | 51.8 | 167.5 | 34.8 | 180.1 | 35.7 | 111.2 | 32.9 | 110.0 |
| | Prophet | 63.5 | 200.8 | 80.0 | 205.2 | 45.7 | 204.7 | 28.6 | 203.7 | 37.2 | 202.8 | 25.0 | 203.8 |
| | Sched | 57.3 | 253.3 | 43.3 | 255.2 | 28.3 | 251.8 | 25.0 | 240.3 | 27.8 | 254.4 | 25.6 | 254.1 |
| | **ZENO (N)** | 72.3 | 96.0 | 78.0 | 93.5 | 50.8 | 107.5 | 35.0 | 111.0 | 38.0 | 93.3 | 28.7 | 77.0 |
| | **ZENO (NB)** | 74.8 | 107.9 | **87.7** | 104.4 | 55.8 | 133.7 | 34.8 | 137.2 | **38.3** | 96.7 | 34.2 | 88.2 |
| | **ZENO (NB) +C** | **75.3** | **56.9** | 84.3$^{tie}$ | **50.3** | 56.5 | **83.3** | 34.3 | **80.6** | 36.0 | **32.2** | 33.5 | **29.9** |
| Dream Instruct | Vanilla | 40.8 | 256.0 | 76.7 | 256.0 | 61.3 | 256.0 | 40.0 | 256.0 | 44.8 | 256.0 | 29.3 | 256.0 |
| | Block | 13.5 | 256.0 | 38.3 | 256.0 | 41.5 | 256.0 | 21.2 | 256.0 | 28.5 | 256.0 | 20.1 | 256.0 |
| | Slow-Fast | 56.0 | 174.9 | 65.7 | 118.0 | **63.8** | 176.6 | **42.6** | 142.6 | 39.5 | 89.7 | 29.9 | 79.3 |
| | KLASS | 75.0 | 140.6 | 83.7 | 119.9 | 63.3 | 180.2 | 42.4 | 181.9 | 48.7 | 78.3 | 46.3 | 78.7 |
| | Prophet | 73.3 | 191.4 | 81.7 | 204.5 | 52.2 | 210.2 | 35.9 | 208.7 | 53.0 | 202.7 | 52.4 | 194.7 |
| | Sched | 40.2 | 255.2 | 74.7 | 252.9 | 15.8 | 206.1 | 5.7 | 132.3 | 40.0 | 252.1 | 26.8 | 250.9 |
| | **ZENO (N)** | 52.5 | 97.6 | 77.7 | 94.5 | 33.0 | 107.3 | 22.6 | 109.3 | **54.2** | 82.7 | 50.0 | 77.7 |
| | **ZENO (NB)** | **79.8** | 110.4 | **87.7** | 102.7 | 56.2 | 136.7 | 37.6 | 136.7 | 54.0 | 85.1 | **54.3** | 83.5 |
| | **ZENO (NB) +C** | 76.3 | **63.0** | 86.3 | **49.5** | 58.0 | **99.4** | 38.6 | **100.5** | 53.3 | **29.7** | 48.8 | **29.3** |
| LLaDA Base | Vanilla | 76.0 | 256 | 81.3 | 256 | 55.7 | 256 | 39.8 | 256 | 42.0 | 256 | 31.1$^{tie}$ | 256 |
| | Block | 61.0 | 256 | 81.3 | 256 | 43.8 | 256 | 29.3 | 256 | 29.8 | 256 | 26.2 | 256 |
| | SlowFast | 76.5 | 94 | 81.7 | 62 | 56.3 | 75 | **41.4** | 104 | 40.5 | 90 | 31.1$^{tie}$ | 52 |
| | KLASS | 75.0 | 109 | 78.3 | 78 | 54.7 | 93 | 40.2 | 127 | 39.8 | 102 | **32.9** | 73 |
| | Prophet | 76.2 | 185 | 81.0 | 176 | 55.7 | 180 | 39.5 | 189 | **42.2** | 180 | 28.7 | 177 |
| | Sched | 76.0 | 221 | 81.3 | 181 | 55.2 | 214 | 39.5 | 227 | 41.2 | 215 | 31.1$^{tie}$ | 204 |
| | **ZENO (N)** | **78.0** | 85 | **83.3**$^{tie}$ | 77 | **58.3** | 87 | 35.2 | **93**$^{tie}$ | 40.8 | 84 | 30.5 | 68 |
| | **ZENO (NB)** | 77.5 | 102 | **83.3**$^{tie}$ | 88 | 56.7 | 95 | 38.3 | 113 | 40.8 | 95 | 31.1$^{tie}$ | 75 |
| | **ZENO (NB) +C** | 77.2 | **76** | **83.3**$^{tie}$ | **58** | 56.2 | **67** | 37.6 | **93**$^{tie}$ | 40.8 | **70** | 30.5 | **41** |
| Dream Base | Vanilla | 74.0 | 256 | 78.7 | 256 | 64.5 | 256 | 44.3 | 256 | **57.2**$^{tie}$ | 256 | 52.4 | 256 |
| | Block | 68.0 | 256 | 75.0 | 256 | 62.8 | 256 | 39.5 | 256 | 53.8 | 256 | **56.1** | 256 |
| | SlowFast | 74.2$^{tie}$ | 129 | 78.3 | 75 | 66.2 | 112 | **46.0** | 115 | 57.0 | 102 | 50.6 | 95 |
| | KLASS | 74.2$^{tie}$ | 145 | 75.7 | 92 | 66.0 | 131 | 43.6 | 137 | 54.5 | 127 | 51.2 | 127 |
| | Prophet | **74.5** | 190 | 78.3 | 179 | **66.3** | 185 | 43.6 | 188 | **57.2**$^{tie}$ | 188 | 53.0$^{tie}$ | 189 |
| | Sched | 74.0 | 236 | 78.7 | 228 | 64.5 | 236 | 44.0 | 231 | 56.5 | 221 | 52.4 | 226 |
| | **ZENO (N)** | 74.0 | **91** | 79.3$^{tie}$ | 82 | 64.3 | 100 | 44.8 | **96** | 54.8 | 89 | 52.4 | 84 |
| | **ZENO (NB)** | 73.8 | 119 | **80.0** | 89 | 66.0 | 117 | 44.3 | 116 | 54.8 | 99 | 52.4 | 97 |
| | **ZENO (NB) +C** | 74.0 | 104 | 79.3$^{tie}$ | **62** | 65.8 | **98** | 44.5 | 99 | 53.5 | **82** | 53.0$^{tie}$ | **77** |

Table 1: Accuracy (%) and NFE table. For instruction-tuned checkpoints, zero-shot performance is noted; for base checkpoints, few-shot prompted performance is shown. ↑ indicates higher is better, ↓ indicates lower is better. **Bold** = best, underline = second-best. $^{tie}$ marker indicates that multiple methods share the same best or second-best value.

| Model | Method | Instruct Acc (%) | Instruct NFE↓ | Instruct s/q↓ | Instruct Tok/s↑ | Instruct Speed-up$_{s/q}$ | Instruct Speed-up$_{Tok/s}$ | Base Acc (%) | Base NFE↓ | Base s/q↓ | Base Tok/s↑ | Base Speed-up$_{s/q}$ | Base Speed-up$_{Tok/s}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaDA | Vanilla | 42.0 | 256 | 12.0 | 13.6 | 1.00× | 1.00× | 54.3 | 256 | 30.8 | 2.6 | 1.00× | 1.00× |
| | Block | 35.1 | 256 | 14.7 | 2.6 | 0.82× | 0.19× | 45.2 | 256 | 49.6 | 1.8 | 0.62× | 0.69× |
| | SlowFast | 53.7 | 123 | 3.3 | 64.5 | 3.63× | 4.74× | **54.6**$^{tie}$ | 80 | 9.9 | 10.1 | 3.10× | 3.88× |
| | KLASS | 51.9 | 138 | 4.5 | 48.0 | 2.67× | 3.53× | 53.5 | 97 | 12.7 | 7.4 | 2.42× | 2.85× |
| | Prophet | 46.7 | 204 | 9.6 | 21.7 | 1.25× | 1.60× | 53.9 | 181 | 21.9 | 3.8 | 1.40× | 1.46× |
| | Sched | 34.6 | 252 | 9.8 | 16.5 | 1.22× | 1.21× | 54.0 | 210 | 25.8 | 3.2 | 1.19× | 1.23× |
| | ZENO (N) | 50.5 | 96 | 3.7 | 55.3 | 3.24× | 4.07× | 54.4 | 82 | 10.1 | 8.9 | 3.04× | 3.42× |
| | ZENO (NB) | **54.2** | 111 | 4.1 | 51.7 | 2.95× | 3.80× | **54.6**$^{tie}$ | 95 | 11.7 | 7.7 | 2.63× | 2.96× |
| | ZENO (NB) +C | 53.3 | **56** | **2.8** | **83.7** | **4.32**× | **6.15**× | 54.3 | **68** | **8.5** | **12.2** | **3.62**× | **4.69**× |
| Dream | Vanilla | 48.8 | 256 | 11.2 | 2.7 | 1.00× | 1.00× | 61.9 | 256 | 23.0 | 2.7 | 1.00× | 1.00× |
| | Block | 27.2 | 256 | 11.4 | 3.3 | 0.98× | 1.22× | 59.2 | 256 | 23.8 | 2.6 | 0.97× | 0.96× |
| | SlowFast | 49.6 | 130 | 3.7 | 23.6 | 3.03× | 8.74× | 62.0 | 105 | 9.6 | 6.6 | 2.40× | 2.44× |
| | KLASS | 59.9 | 130 | 4.4 | 32.5 | 2.55× | 12.04× | 60.9 | 126 | 12.3 | 5.1 | 1.87× | 1.89× |
| | Prophet | 58.1 | 202 | 8.9 | 18.0 | 1.26× | 6.67× | **62.1** | 186 | 16.8 | 3.6 | 1.37× | 1.33× |
| | Sched | 33.9 | 225 | 8.7 | 3.3 | 1.29× | 1.22× | 61.7 | 230 | 20.9 | 3.0 | 1.10× | 1.11× |
| | ZENO (N) | 48.3 | 95 | 3.6 | 38.6 | 3.12× | 14.30× | 61.6 | 90 | 8.3 | 7.4 | 2.78× | 2.74× |
| | ZENO (NB) | **61.6** | 109 | 4.0 | 35.7 | 2.80× | 13.22× | 61.9 | 106 | 9.8 | 6.4 | 2.35× | 2.37× |
| | ZENO (NB) +C | 60.2 | **62** | **2.8** | **54.5** | **3.96**× | **20.19**× | 61.7 | **87** | **8.1** | **8.0** | **2.84**× | **2.96**× |

Table 2: Efficiency table capturing the remaining aspects of Table 1. Average accuracy over all six datasets, NFE, s/q (seconds per question), Tok/s, and both speed-up figures are noted here. **Speed-up**$_{s/q}$ is Vanilla's wall-clock time per query divided by the method's; **Speed-up**$_{Tok/s}$ is the method's throughput relative to Vanilla's. Here, **Bold** represents best and underline represents second-best.

### 5.2 Efficiency

As shown in Table 2, ZENO offers a favorable accuracy–efficiency trade-off across latency, throughput and generation quality. ZENO(NB)+C has the lowest NFE and the lowest per-query inference time of all methods on every checkpoint. The best quality-efficiency trade-offs come from two configurations. For quality-critical regimes, ZENO(NB) delivers the best accuracy with high throughput, leading all competitive methods with $54.2\%$ average accuracy (a $+12.2$ point improvement over Vanilla) on LLaDA Instruct and $61.6\%$ on Dream Instruct, while still yielding solid per-query speedups of $2.95\times$ on LLaDA Instruct and $2.80\times$ on Dream Instruct and the token-throughput gains of $3.80\times$ and $13.22\times$ respectively. On the other hand, when wall-clock latency is paramount, ZENO(NB)+C achieves remarkable acceleration by confidence-guided early reveal, reaching $\sim 4\times$ per-query speedup ($54.5$ Tok/s, $\sim 20.2\times$ throughput) on Dream Instruct and a $\sim 4.3\times$ per-query speedup (83.7 Tok/s, $6.15\times$ throughput) on LLaDA Instruct at a record-low $56 - 62$ NFE range. Although ZENO(NB)+C yields substantially higher throughput and efficiency, this comes with a minor trade-off in accuracy compared to ZENO(NB) (e.g., $53.3\%$ vs. $54.2\%$ on LLaDA Instruct and $60.2\%$ vs. $61.6\%$ on Dream Instruct). Together, both variants dismantle the conventional compromise between diffusion sampling speed and downstream task fidelity. On DiffusionGemma (Team et al. 2026) with in-distribution and held out test sets, we achieve $1.3\times$ average inference speedup and around $0.2$ points average accuracy gain with ZENO. Appendix F.2; Table 5 details the results.

### 5.3 Ablation Studies

**Results With KV Cache Enabled.** Because ZENO never re-masks or modifies tokens once they are generated, committed positions retain their approximate key/value representations for the remainder of the decoding phase. This structural property enables prefix KV-caching across subsequent denoising steps. We compare the decoding latency and accuracy with and without KV-caching across the base and instruct model variants of LLaDA and Dream. Caching universally improves decoding throughput across all settings. Compared with the vanilla and no-KV cache variants of ZENO, the setup for LLaDA Instruct and Base model achieves around $5.5\times$ (Figure 3) and $7.9\times$ per-query latency ($\sim 7.2\times$ and $8.7\times$ higher token throughput) versus the vanilla baseline (Table F.1). For Dream Instruct, ZENO(NB)+C with KV cache cuts per-query latency by $5.6\times$ (throughput up by $26.2\times$), with accuracy improving from $48.8\%$ to $54.9\%$ (+6.1 points). For Base models under the few-shot setup, ZENO(NB) with KV gives a $6.4\times$ per-query speedup, with a $0.3$ points accuracy gain on LLaDA and for Dream a $6.2\times$ speedup with $1.8$ points accuracy drop. Detailed table and experimental results are provided in Appendix F.1.

**Generation Length and Step Budget.** To ensure token-window length and step budget invariance of our method, we perform experiments with different token generation length window ($L$) and step budget ($S$) with both LLaDA and Dream Instruct models. We experiment with three

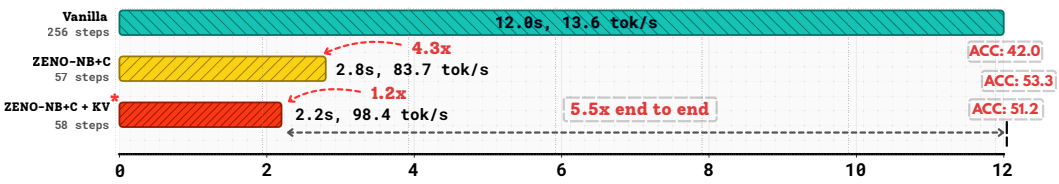



Figure 3: The efficiency comparison between Vanilla vs ZENO(NB)+C and ZENO(NB)+C with KV cache. The efficiency gain is computed against average time of inference per question.

configurations: 256 tokens + 256 steps; 512 tokens + 256 steps and 512 tokens + 512 steps. We compare all the ZENO variants with SlowFast (Our closest baseline), Vanilla and Block diffusion sampler. Experimentally, we validate that, unlike the other methods, with bigger generation window and more denoising steps ZENO actually improves the accuracy of the models. For both model classes, as sequence lengths expand, baseline methods frequently suffer accuracy drops (e.g., Vanilla drops from 48.8% to 42.6% mean accuracy on Dream). In contrast, ZENO reaches its best accuracy at the largest budget, with ZENO(NB) reaching a peak mean accuracy of 56.0% and 62.6%; and ZENO(NB)+C reaching a peak of 55.4% and 61.4% on LLaDA and Dream respectively at 512 tokens — 512 steps configuration. ZENO achieves this while maintaining drastically lower NFE than all the baseline sampling methods. Detailed results are provided in Appendix F.3.

**Effect of Changing Growth Base.** As explained in Eq. 3, the growth base $\gamma$ determines the expansion rate of the active decoding window. We vary $\gamma$ for four different values across all three of ZENO variants. Varying $\gamma \in \{2, 3, 4, 5\}$ demonstrates that our primary variants, ZENO(NB) and ZENO(NB)+C, exhibit sustained stability across both LLaDA and Dream, with mean accuracy fluctuating by at most 0.5 points across all rates on LLaDA. In contrast, ZENO(N) is notably very sensitive to $\gamma$. Setting $\gamma = 2$ provides the most reliable balance across all architectures and individual benchmark tasks. It avoids prematurely exposing unconditioned future positions while allowing subsequent block-advancement and confidence caching to drive high inference throughput. Appendix F.5 contains the detailed experimental results and discussion.

**Effect of Different Post Switch Block Length After the Switch Width.** Once the exponential dynamic window reaches the switch width ($W^\star$), decoding progresses in fixed blocks. As shown in Appendix F.6 (Table 10), varying the fixed block length across 16, 32, 64, 128 reveals a trend in accuracy vs efficiency tradeoff. From this experiment, it is clearly evident that inference throughput increases with the increase in block length, but at the cost of degraded accuracy. Particularly on Dream, where expanding from 64 to 128 tokens causes a steep drop in mean accuracy (e.g., $59.7\% \rightarrow 50.9\%$). A block length of 64 strikes the optimal balance across both LLaDA and Dream, capturing substantial throughput gains while preserving peak task accuracy. Detailed experimental results and tables are provided in Appendix F.6.

**Effect of Different Confidence Thresholds** Confidence threshold governs the trade-off between decoding quality and decoding efficiency. In this setup, we set the confidence threshold above which the sampler is free to unmask multiple tokens on the basis of confidence within the given block. To trace the optimality frontier, we experiment with three confidence thresholds 0.6, 0.8 and 0.9 across both base and instruction-tuned variants of LLaDA and Dream. NFE increases monotonically with the threshold, but accuracy does not. Increasing the threshold from 0.6 to 0.8 boosts accuracy by 5.0 points on LLaDA Instruct and 8.8 points on Dream Instruct. Increasing it to 0.9 lowers mean accuracy to 52.8% and 58.4%, respectively, while further increasing NFE. We therefore use $\tau = 0.8$ as the default setting. Detailed results are provided in Appendix F.7.

**Robustness in non-deterministic settings.** For robust evaluation and reproducibility, we use zero temperature with a confidence-based greedy sampling setup for generating deterministic outputs. But in practical deployment settings, higher temperature with stochastic sampling methods is employed. To verify that our gains are not artifacts of deterministic search, we evaluate stochastic decoding across five independent random seeds 42, 123, 512, 1024, 4096 at temperature $\tau_{\text{gen}} = 0.7$. Our experiments reveal that the performance advantages established in deterministic settings strongly hold under stochastic temperature sampling for both ZENO(NB) and ZENO(NB)+C samplers. Our method consistently surpasses Vanilla decoding across model families, achieving

**Comparative Analysis: Robust Mathematical Reasoning**

**Q:** Katarina has 5 less cookies than Max. Max has 12 more cookies than the Cookie Monster and Summer has 23 more than Max. If Katarina has 68 cookies, how many cookies do they have in total?

**Vanilla**

To determine the total number of cookies ...
**Step 1:** Determine the number of cookies Max has. We know that Katarina has 68 .... Therefore, Max has: $68 + 5 = 83$ ×....
**Step 3:** ... Therefore, Summer has: $83 + 23 = 96$ ×...
Therefore, the total number of cookies they have is $\boxed{247}$.

**ZENO (Ours)**

To determine the total number of cookies ...
**Step 1:** Determine the number of cookies Max has. Katarina has 68 cookies.... Therefore, Max has: $68 + 5 = 73$ ✓...
**Step 3:** ... Therefore, Summer has: $73 + 23 = 96$ ✓...
Thus, the total number of cookies they have is $\boxed{298}$.

Figure 4: The **Vanilla** setup (left) fails at multiple instances while generating the trajectory on the other hand, ZENO's directional decoding gives it superior mathematical reasoning and correctness.

average gains of $+11.4$ and $+10.5$ points on LLaDA, and $+9.4$ and $+8.6$ points on Dream for ZENO(NB) and ZENO(NB)+C samplers, respectively. Furthermore, the seed-to-seed standard deviation remains tightly bounded within $\leq 0.8$ points, confirming that the margins separating ZENO variants from the baselines vastly exceed sampling variance and reflect genuine improvements. Appendix F.8 contains the detailed experimental validation results.

**Output Analysis:** Figure 4 shows an example where the directional decoding helps in proper mathematical reasoning. Appendix I provides detailed output statistics and trajectory analysis (Figure 14, 15, 16). Further entropy and stability analysis revealed that ZENO produces far more stable and confident output throughout the generation steps Figure 10. This further strengthens the trajectory stability claims. Figure 11, 12 and 13 provides more conclusive proof of ZENO's substantial gain in performance.

**Statistical Significance.** To verify whether the accuracy margins over Vanilla decoding reflect genuine algorithmic improvements rather than seed variation, we conduct two-tailed paired $t$-tests on the composite six-benchmark mean accuracy matched by seed. Across both base architectures, every comparison against Vanilla is statistically significant well beyond $\alpha = 0.001$: on LLaDA, ZENO(N), ZENO(NB), and ZENO(NB)+C exhibit large, reliable gains ($p = 5.0 \times 10^{-5}$, $p = 1.0 \times 10^{-6}$, and $p = 1.5 \times 10^{-5}$, respectively). On Dream, both ZENO(NB) and ZENO(NB)+C similarly demonstrate decisive improvements ($p = 5.0 \times 10^{-6}$ and $p = 2.0 \times 10^{-5}$).

## 6 Conclusion

In this work, we introduce **ZENO** (**Z**one **E**xpanding decodi**N**g **O**perator), a training-free, inference-time geometric scheduler for discrete diffusion models. ZENO class of dynamic schedulers addresses two fundamental bottlenecks of discrete diffusion models. Firstly, the lack of directional prior resulting in premature commitment to wrong outputs without reasoning, and secondly, the efficiency gap. Rather than treating sequential decoding and bidirectional diffusion as mutually exclusive paradigms, ZENO dynamically transitions from semi-autoregressive prefix grounding to parallel diffusion refinement within a single, continuous generation instance at inference time. This change drastically improves the model's performance by a solid margin. ZENO consistently outperforms the current SOTA samplers by a solid margin in both accuracy and efficiency across most benchmark tasks, without any form of task-specific fine-tuning. As a dynamic sampler ZENO is a plug-and-play solution. We propose ZENO(NB)+C as our best-performing sampler, with the optimal accuracy-efficiency trade-off. On the performance side, on LLaDA and Dream, ZENO(NB) raises the accuracy by $12.2 - 12.8$ point on Instruction tuned models, while ZENO(NB)+C cuts NFE by $66 - 78\%$ and reduces per query latency by up to $4.3\times$, reaching $7.9\times$ with KV caching. The gains transfer without tuning to DiffusionGemma, including in and out of distribution benchmarks. ZENO performs strongest on reasoning and coding benchmarks and trails slightly on memorization/knowledge-intensive tasks. In future combining ZENO with remasking and editing techniques is a natural next step to further improve accuracy.

## ETHICS STATEMENT

This study adheres fully to the ICLR Code of Ethics throughout the research design, experimentation and manuscript preparation. All models and datasets used for evaluation are publicly available and used solely for academic research; we do not make any kind of ownership or licensing claims over any of them. All data remains subject to the original licenses, terms of use, and copyright protections established by the respective source providers. We have made every effort to ensure our usage complies with these original terms. Through ZENO, we introduce a training-free, inference-time decoding strategy: beyond adding new capabilities by accelerating and improving the sampling process of existing diffusion language models it does not alter what a model can generate. As such, we do not anticipate misuse risks materially different from those already associated with the underlying base models' capabilities.

## REPRODUCIBILITY STATEMENT

We take reproducibility very seriously and have aimed to make every result in this paper independently verifiable. ZENO is a training-free, inference-time decoding method with no learned parameters (Section 3), so there is no training pipeline, checkpoint, or optimizer state to reproduce – only the deterministic decoding procedure itself, given a fixed pretrained model.

**Determinism and seeding.** Our primary results (Tables 1–4) use greedy decoding (temperature $\tau_{\text{gen}} = 0$) with a fixed seed of 42, which is fully deterministic given the same model weights, hardware, and floating-point settings. To additionally verify that our gains are not an artifact of this single deterministic configuration, we separately report a five-seed stochastic study at temperature 0.7 (seeds 42, 123, 512, 1024, 4096; Table 13), including per-seed results and standard deviations, together with paired significance tests against Vanilla decoding (Section 5.3).

**Data and models.** All four evaluated checkpoints (LLaDA-8B-Base/Instruct, Dream-7B-Base/Instruct) and all six benchmarks (GSM8K, SVAMP, MMLU, MMLU-Pro, MBPP, HumanEval, GSM-Plus, Math-500) are publicly released by their original authors; no data was collected or modified by us beyond standard prompt formatting, which is disclosed in full in Appendix D, including the exact system prompts (Figure 9) and few-shot example counts per dataset (Table 3).

**Hyperparameters and Computational Statistics.** We disclose all the hyperparameters used throughout the paper and provide a combined hyperparameter table in Section J Table 16. The detailed computational statistics are provided in Section G; Table 12.

**Metrics and measurement protocol.** Accuracy, NFE, tokens/second, and seconds/question are each defined precisely in Appendix C and D, including how NFE is counted from recorded decoding trajectories (rather than assumed from the nominal step budget) and how Tok/s and s/q are measured around the sampler call, excluding tokenization, answer extraction and unit-test runtime.

**Code and artifact release.** Upon acceptance, we will publicly release our full codebase, evaluation harness, and the raw per-question decoding traces (including token-by-token reveal order) underlying every table in this paper, enabling exact reproducibility of all reported numbers without re-running inference.

## AI USE STATEMENT

In accordance with AI usage guidelines, generative AI tools were only used for the purposes of paraphrasing, grammar correction, and spell-checking. Generative AI tools were also used to provide assistance with coding and implementation. All AI-generated suggestions and code were carefully reviewed and fully verified. We take the full responsibility for the final content, implementation, and results presented in this work.

## REFERENCES

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, 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023a. URL https://arxiv.org/abs/2107.03006.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023b. URL https://arxiv.org/abs/2107.03006.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a", 2024. URL https://arxiv.org/abs/2309.12288.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models, 2022. URL https://arxiv.org/abs/2205.14987.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

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, 2021b. URL https://arxiv.org/abs/2103.03874.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020a. URL https://arxiv.org/abs/2006.11239.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020b. URL https://arxiv.org/abs/2006.11239.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models, 2022. URL https://arxiv.org/abs/2110.02037.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. Klass: Kl-guided fast inference in masked diffusion models, 2026. URL https://arxiv.org/abs/2511.05664.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding, 2026. URL https://arxiv.org/abs/2508.19982.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. 2024.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.

Amr Mohamed, Yang Zhang, Michalis Vazirgiannis, and Guokan Shang. Fast-decoding diffusion language models via progress-aware confidence schedules, 2025. URL https://arxiv.org/abs/2512.02892.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan LI. Large language diffusion models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems?, 2021. URL https://arxiv.org/abs/2103.07191.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models, 2024. URL https://arxiv.org/abs/2406.07524.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data, 2025. URL https://arxiv.org/abs/2406.04329.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021. URL https://arxiv.org/abs/2011.13456.

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

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Puyu Zeng, Yuxuan Wang, Biqing Qi, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with slowfast sampling: The three golden principles, 2026. URL https://arxiv.org/abs/2506.10848.

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, 2025. URL https://arxiv.org/abs/2505.22618.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025. URL https://arxiv.org/abs/2508.15487.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12), May 2026. ISSN 2095-2236. doi: 10.1007/s11704-026-60308-3. URL http://dx.doi.org/10.1007/s11704-026-60308-3.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dllm: Simple diffusion language modeling, 2026. URL https://arxiv.org/abs/2602.22661.

## Extended Results and Ablations

## Contents of this Appendix

**A** Random Walk: A Thought Experiment for Intuitive Understanding. ..... 14
**B** ZENO: Some Derivations ..... 16
**C** Metric Definitions ..... 16
**D** Dataset, Prompts and Evaluation ..... 17
**E** Measurement Protocol ..... 18
**F** Extended Results and Ablation ..... 18

- F.1 Results with KV-Cache Enabled ..... 18
- F.2 Applicability to Other Model Sizes and Datasets ..... 19
- F.3 Generation Length and Step Budget ..... 20
- F.4 Few-Shot Prompting on Instruction-Tuned Checkpoints ..... 21
- F.5 Growth Base: How Fast Should the Window Widen? ..... 21
- F.6 Block Length After Switch Width ..... 21
- F.7 Confidence Threshold ..... 22
- F.8 Sampled Decoding at Temperature 0.7 ..... 23

**G** Computational Statistics ..... 23
**H** Algorithm ..... 25
**I** Output Analysis ..... 26
**J** Hyperparameter Table ..... 28
**K** Nomenclature ..... 28

# A A THOUGHT EXPERIMENT FOR INTUITIVE UNDERSTANDING.

To establish the role of directional prior we perform the random walk experiment in higher dimensions as a philosophical thought experiment for gaining the intuition behind **ZENO**. We study a random walker traveling from a starting point $A$ to a distant target $B$ in $\rho$ dimensions, with and without a bias towards the final point $B$. Through this experiment, we try to show: **"How much does a directional prior help, and how does its value change as the search space grows in dimension?"**

**Experimental Setup.** Let $\rho \in \{1, \ldots, 5\}$, $A = \mathbf{0}$ and $B$ be at any distance $D$. At each of $S$ discrete steps, one axis $a_t \sim \text{Uniform}\{0, \ldots, \rho - 1\}$ is selected, and the walker takes a unit step $\pm 1$ along it. Along the target axis ($a_t = 0$) the step is $+1$ with probability $v$ and $-1$ with probability $1 - v$. All other axes remain unbiased ($v = 0.5$). Thus $v = 0.5$ recovers the symmetric (prior-free) random walk, while $v = 1$ corresponds to a maximally confident prior that always moves toward $B$ whenever the target axis is selected. A walker is counted as a *hit* if it ever comes within radius $R$ of the target, i.e. $\min_{t \le T_{max}} \|x_t - B\|_2 \le R$. For each dimension and each bias level, we simulate 20000 independent walks per ($\rho$,p) cell.

## A.1 EXPERIMENTS IN 2D AND 3D

**2D Unbiased vs Biased Random Walk.** In Figure 5 the first column represents an unbiased random walk which has an equal probability ($v = 0.5$) of moving about any two consecutive direction. As seen in the figure, it has a really low probability of reaching the desired destination point B. But as we move along the columns we increase the bias from 0.6 to 1 and the probability of reaching the target increases drastically.

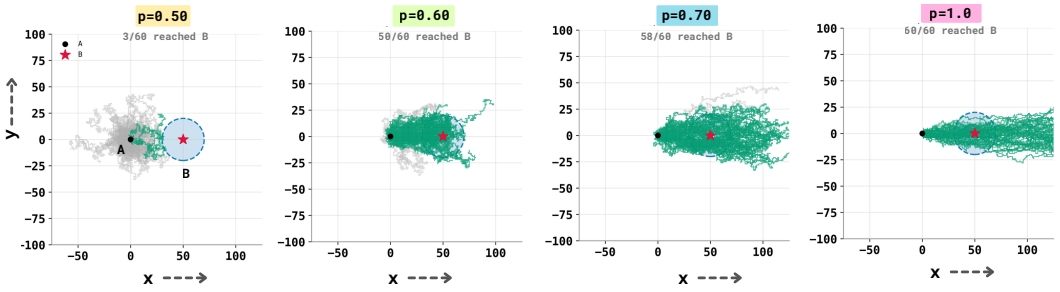



Figure 5: Hit rate with Unbiased vs Biased 2D Random walk.

**3D Unbiased vs Biased Random Walk.** Figure 6 demonstrates the 3D random walk. In this experiment, another dimension is added which exponentially increases the search space. Here for the unbiased walk, the first column shows the hit rate of an unbiased walk where there is equal probability of moving up or down; left or right and forward or backward. Moving along the columns as the bias increases the probability of reaching the target point zone also increase although the search complexity increases.

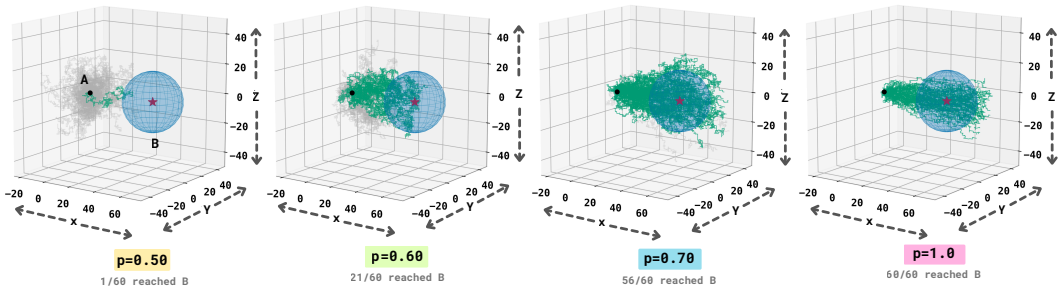



Figure 6: Hit rate with Unbiased vs Biased 3D Random walk.

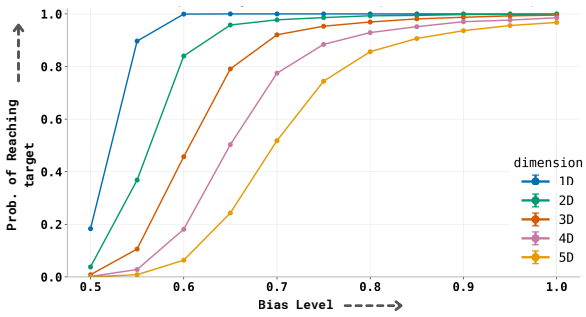



Figure 7: Probability of reaching the target vs bias levels in multiple dimensions.

### A.2 Expanding experiments from 1D to 5D

To show the effect of biased prior in higher dimensions, we run a full-scale experiment from 1D expanding to 5D with the same setup of increasing bias along each dimension, and note the probability of reaching or hitting the target zone.

**Without a prior, search collapses with dimension.** The unbiased walker ($v = 0.5$) reaches the target with probability $18.3\%$ in 1D, but this drops to $3.8\%$ in 2D, $0.83\%$ in 3D, $0.17\%$ in 4D, and only $0.045\%$ in 5D. This is a roughly $400\times$ decline over only four added dimensions. The cause is the growing space geometry. An undirected/unbiased walker spreads its step budget over an exponentially growing volume, making the probability of reaching a fixed-size target region vanish really quickly. The trajectory plots in (Figure 5 and 6) make this concrete. At $v = 0.5$ the paths form a cloud around the starting point and only $3/60$ (2D) and $1/60$ (3D) ever touch the hit region.

**With a prior, the target remains reachable.** Introducing even a moderate amount of bias changes the behavior completely. At $v = 0.70$ the hit probability for 1D is $100\%$, $97.8\%$ for 2D, $92.1\%$ for 3D, $77.5\%$ for 4D and $51.8\%$ in 5D. This shows that a biased walker succeeds in majority of the cases where the unbiased walker fails. In Figure 7 along every dimension, the curves increase monotonically as bias increases. In the trajectories shown in Figure 5 and 6 it is visible that the trajectory cloud gets redirected and shrinks towards the target point. At $v = 0.60$, $50/60$ (2D) walkers already reach the target and at $v = 1.00$ all walks reach the target for both 2D and 3D cases.

**Correlating with `ZENO's` directional flow.** For diffusion language models the intuitive analogy comes from this analysis. Where the model tries to unmask tokens from the generation window (e.g., 256 token window). As the model tries to generate tokens from anywhere on the canvas, it gives a very noisy prior for the next denoising step, which if we think lives in $\mathbb{R}^{256}$, becomes a point in that reasoning space. Now there is no guarantee of this point being stable as the dimension is really large. This is where `ZENO's` directional generation philosophy comes into the picture. We hypothesize that starting from a small degree of freedom helps in providing well-informed, biased priors, which results in better reasoning and generation quality.

**Key Takeaways**

- **Undirected search fails in high dimensions.** As dimension increases, the probability of reaching a distant target by chance decreases rapidly.
- **Directional prior counteracts this decay.** It concentrates the limited step budget towards the actual direction, keeping the target reachable where unbiased search almost always fails.
- **The benefit scales with dimension.** The relative advantage of the prior grows as the dimension increases, and higher dimensions requiring stronger prior.
- **Strong priors are essential, not optional.** A strong prior toward the goal is not a minor refinement but the component that makes high-dimensional search tractable.

# B ZENO: SOME DERIVATIONS

This appendix supports Section 3.2. Appendix B.1 derives the phase count and the increment growth of the zone schedule; Appendix H gives the decoding procedure in full.

## B.1 ZONE SCHEDULE

**Number of Phases.** Ignoring rounding, the geometric schedule of Eq. 3 has $w_m = w_0\gamma^{(m-1)(\kappa+1)}$, so the first width to reach the generation length satisfies

$$w_0\,\gamma^{(m-1)(\kappa+1)} \;\geq\; L \quad\Longleftrightarrow\quad m \;\geq\; 1 + \frac{\log_\gamma(L/w_0)}{\kappa+1}, \tag{8}$$

and since that width is clamped to $L$ and terminates the schedule,

$$K \;=\; \left\lceil \frac{\log_\gamma(L/w_0)}{\kappa+1} \right\rceil + 1 \;=\; \Theta\Big(\frac{\log L}{\kappa+1}\Big). \tag{9}$$

The phase count is logarithmic in the generation length, which is what keeps the grounding stage affordable: the narrow zones are traversed in a handful of phases rather than paid for across the whole sequence. Larger $\kappa$ thins the schedule further, trading a finer ramp for fewer phases.

**Growth of the Increment.** For $2 \leq m \leq K-1$, again ignoring rounding,

$$\Delta_m \;=\; w_m - w_{m-1} \;=\; w_{m-1}\big(\gamma^{\kappa+1} - 1\big), \tag{10}$$

so the number of positions a phase must ground is proportional to the width already settled: the increment grows at the same rate as the zone. The last phase is the extreme case. Since $w_{K-1} < L \leq \gamma^{\kappa+1} w_{K-1}$, the clamp gives

$$\Delta_K \;=\; L - w_{K-1} \;\leq\; L\big(1 - \gamma^{-(\kappa+1)}\big), \tag{11}$$

which for the default $\gamma = 2$, $\kappa = 0$ is half the generation region opened in a single phase (128 of 256 positions). Two effects compound here: the widest zone is also the one whose quota $k_t$ in Eq. 6 is largest, so the phase with the weakest ordering constraint is also the phase unmasking the most positions per pass. The constant branch of Eq. 5 exists to remove this, bounding $\Delta_m \leq \beta$ for every phase past $W^\star$ while leaving the geometric ramp below $W^\star$ untouched.

# C METRIC DEFINITIONS

Four quantities are reported throughout the paper capturing exact model performance, speed and efficiency metrics. Here we detail the way we compute and report the numbers in the paper.

**Accuracy.** Exact-match correctness on the benchmark's own answer key, computed per question and averaged over the evaluated subset. For the mathematics benchmarks the final answer is extracted from inside `\boxed{}`; for the multiple-choice benchmarks the predicted option letter is extracted and compared against the gold answer; for the code benchmarks a generation is correct if the extracted function passes every unit test associated with the problem. For all the datasets across the experiments we note `pass@1` performance.

**NFE (Number of Function Evaluations).** NFE is the number of forward passes of the model $f_\theta$ spent on decoding one question, counted as in Algorithm 1: every call of $f_\theta$ counts, including the final cleanup pass, whether or not it commits a token, while steps with a zero quota make no call and are not counted. Defined this way, NFE does not depend on the batch size; in batched runs we count only the passes in which the question itself is still being decoded, excluding passes in which it waits for other questions in the batch. For fixed-schedule samplers NFE equals the step budget (256 for Vanilla and Block); for adaptive samplers it varies by question. We average NFE over the questions of each benchmark and report the mean of the six per-benchmark values.

**Tokens Per Second (TPS).** Here we store the number of tokens the model actually generated, tokenized with that model's own tokenizer, divided by the measured decode time:

$$\text{TPS} = \frac{\sum_q \left|\,\text{tokenize}\big(y_q\big)\,\right|}{\sum_q \Delta t_q}, \tag{12}$$

where $y_q$ is the generation for question $q$ and $\Delta t_q$ its measured decode time. All TPS numbers are computed from the per-question records via Eq. 12 and then averaged over the dataset and reported in the paper.

**Seconds Per Question (s/q).** This is the wall-clock decode time per question, measured around the sampler. Prompt tokenization, answer extraction, code unit test runtime, answer verification are excluded from this. Here, s/q reflects generation cost and not the cost of grading it. This is also computed per sample and then averaged over the dataset for reporting.

## D Dataset, Prompts and Evaluation

We evaluate on six benchmarks spanning three task families: (i) mathematical reasoning, (ii) general knowledge and (iii) code generation. We choose this diverse set to make the comparison not dominated by any single QA format. Mathematics and knowledge tasks have exactly checkable answers, whereas code generation requires a syntactic check where there is ground-truth code and set of test cases. We validate the code accuracy by individually compiling both the ground-truth code and the generated code and comparing the outputs or success rate. The examples for type of questions encountered in each dataset are shown in Figure 8

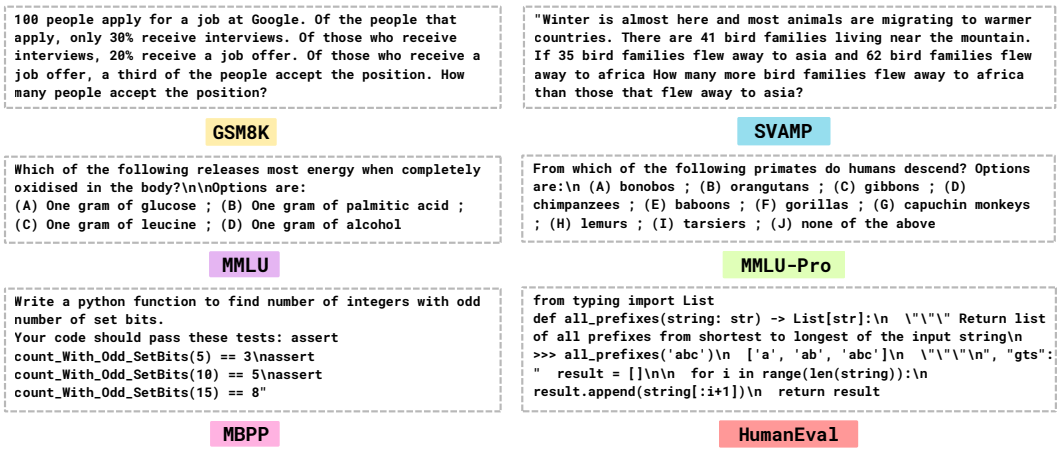



Figure 8: A sample from each of the datasets (GSM8K, SVAMP, MMLU, MMLU-Pro, MBPP and HumanEval) used for evaluating all the methods and models.

**System prompts.** Figure 9 contains the prompts used for all the datasets for the instruct model experiments. For the base model experiments we use few-shot prompts and disclose the number of examples used for each dataset in Table 3.

**Evaluation statistics.** For evaluating all the methods across four model checkpoints, we choose a total of 2284 questions selected randomly across the six datasets. Table 3 contains the dataset-wise breakdown of the number of questions selected per dataset. We choose the subset cumulatively to balance statistical significance with computational cost. Across all our experiments including main studies and ablation experiments we perform over **600K** output evaluations.

**Data description table.** Table 3 contains the details of number of questions considered per dataset, containing subject, a brief description of the dataset and the prompting strategy for evaluation.

**GSM8K - SVAMP**

```
You are a helpful assistant. Solve this question and give the final answer inside \boxed{answer}.
```

**MMLU - MMLUPro**

```
You are a helpful assistant. Solve the problem carefully and provide the final answer as the option
letter. End your response with: Correct Option is: \boxed{}
```

**MBPP - HumanEval**

```
You are a helpful assistant. Write the complete Python function that solves this problem. Reply with
only the code inside a single ```python fenced block.
```

Figure 9: System prompts used for Instruction tuned models across all evaluation setups.

| Dataset | Datapoints | Subjects | Task Type | Prompting (n-shot) | |
|---|---|---|---|---|---|
| | | | | Base | Instruct |
| GSM8K | 400 | 1 | Grade-school level math word problems | 5 | 0 |
| SVAMP | 300 | 1 | Elementary arithmetic word problems | 3 | 0 |
| MMLU | 600 | 20 | MCQ knowledge, 4 options (30/subject) | 5 | 0 |
| MMLU-Pro | 420 | 14 | MCQ knowledge, 10 options (30/subject) | 5 | 0 |
| MBPP | 400 | 1 | Code generation (basic programming) | 3 | 0 |
| HumanEval | 164 | 1 | Code generation (function completion) | 3 | 0 |

Table 3: Table containing evaluation benchmarks: Dataset statistics (Questions selected per dataset for evaluation and containing subjects) and prompting configuration for Base and Instruct models.

## E Measurement Protocols

**Batch Size and Timing.** For dLLMs evaluation one observation is the wall-clock time and the token/s numbers are very different for single query processing and batch query processing. The reason is attributed to diffusion models pre-allocating memory and parallel token generation. On the other hand, NFE and the Accuracy parameters are independent of batch size. For all our experiments, we keep the batch size as 1 while generating responses. This choice is made to report the numbers as accurately as possible. We store per-question metrics and note the average numbers over the datasets individually.

**Question Sets and Prompting.** For all the experiments, every method sees an identical, fixed question set per benchmark (Appendix D). We use the exact same prompt for all the methods and across model classes (both base and instruct-tuned variants) to ensure a controlled testbed and eliminate experimental bias. For the base models, we use few-shot prompts and for instruction-tuned models we use zero-shot prompting for evaluation. Notably, prior studies in different papers predominantly restrict their evaluations to few-shot settings on a single model variant (either on Base or Instruct variant). In contrast we demonstrate our method generalizes across base and instruct model paradigms, achieving state-of-the-art (SOTA) performance in both zero-shot and few-shot regimes.

## F Extended Results and Ablations

### F.1 Results with KV-Cache Enabled

Table 4 reports accuracy and decoding cost for each of ZENO's variants with Fast-dLLM (Wu et al. 2025) KV cache enabled vs. without KV cache enabled setup. ZENO is KV-enabled because it does not re-mask or edit tokens already generated, enabling a committed position to retain its key/value entries for the remainder of the phase. This makes the prefix cache reusable across substeps. We perform evaluation on both base and instruction-tuned checkpoints of LLaDA and Dream model families. From Table 4 it is evident that caching improves the throughput in all the enabled settings, giving the best observed per query speedup jump from $4.0\times$ to $5.6\times$ for Dream Instruct (token

| Model | Method | Instruct Acc (%) | Instruct s/q↓ | Instruct Tok/s↑ | Instruct Speed-up$_{s/q}$ | Instruct Speed-up$_{Tok/s}$ | Base Acc (%) | Base s/q↓ | Base Tok/s↑ | Base Speed-up$_{s/q}$ | Base Speed-up$_{Tok/s}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaDA | Vanilla | 42.0 | 12.0 | 13.6 | 1.00× | 1.00× | 54.3 | 30.8 | 2.6 | 1.00× | 1.00× |
| | Block | 35.1 | 14.7 | 2.6 | 0.82× | 0.19× | 45.2 | 49.6 | 1.8 | 0.62× | 0.69× |
| | ZENO (N) | 50.5 | 3.7 | 55.3 | 3.24× | 4.07× | 54.4 | 10.1 | 8.9 | 3.04× | 3.42× |
| | ZENO (NB) | **54.2** | 4.1 | 51.7 | 2.95× | 3.80× | **54.6** | 11.7 | 7.7 | 2.63× | 2.96× |
| | ZENO (NB) +C | 53.3 | 2.8 | 83.7 | 4.32× | 6.15× | 54.3 | 8.5 | 12.2 | 3.62× | 4.69× |
| | ZENO (N) with KV | 48.2 | 2.9 | 66.3 | 4.14× | 4.88× | 53.8 | 4.1 | 19.9 | 7.51× | 7.65× |
| | ZENO (NB) with KV | 52.1 | 3.2 | 60.4 | 3.75× | 4.44× | **54.6** | 4.8 | 17.0 | 6.42× | 6.54× |
| | ZENO (NB) +C with KV | 51.2 | **2.2** | **98.4** | **5.45**× | **7.24**× | 54.3 | **3.9** | **22.6** | **7.90**× | **8.69**× |
| Dream | Vanilla | 48.8 | 11.2 | 2.7 | 1.00× | 1.00× | 61.8 | 23.0 | 2.7 | 1.00× | 1.00× |
| | Block | 27.2 | 11.4 | 3.3 | 0.98× | 1.22× | 59.2 | 23.8 | 2.6 | 0.97× | 0.96× |
| | ZENO (N) | 48.3 | 3.6 | 38.6 | 3.12× | 14.30× | 61.6 | 8.3 | 7.4 | 2.78× | 2.74× |
| | ZENO (NB) | **61.6** | 4.0 | 35.7 | 2.80× | 13.22× | **61.9** | 9.8 | 6.4 | 2.35× | 2.37× |
| | ZENO (NB) +C | 60.2 | 2.8 | 54.5 | 3.96× | 20.19× | 61.7 | 8.1 | 8.0 | 2.84× | 2.96× |
| | ZENO (N) with KV | 44.5 | 2.4 | 54.5 | 4.67× | 20.19× | 59.6 | **3.2** | **17.8** | **7.19**× | **6.59**× |
| | ZENO (NB) with KV | 55.0 | 2.8 | 47.8 | 4.00× | 17.70× | 60.1 | 3.7 | 15.2 | 6.22× | 5.63× |
| | ZENO (NB) +C with KV | 54.9 | **2.0** | **70.8** | **5.60**× | **26.22**× | 59.7 | 3.5 | 16.3 | 6.57× | 6.04× |

Table 4: Accuracy and decoding cost of ZENO with and without KV-cache enabled on LLaDA and Dream. Instruct results are zero-shot and Base model results are few-shot prompted. Tok/s is true throughput: tokens actually generated, divided by measured decode time. **Speed-up**$_{s/q}$ is Vanilla's wall-clock time per query divided by the method's s/q and **Speed-up**$_{Tok/s}$ is the method's Tok/s relative to Vanilla's, on the same checkpoint. Vanilla has no cached counterpart. **Bold** indicates best, underline indicates second-best.

throughput climbing from 20.19× to 26.22×) and from 3.62× to 7.90× a 118% gain for LLaDA Base model, while losing little to no amount of generation accuracy for our main ZENO(NB)+C setup.

**The cache is an approximation.** Attention in a masked diffusion transformer is bidirectional, so a committed position's key/value entries still depend on the masked positions to its right, which keep changing as the window fills. Freezing the prefix cache at a phase boundary therefore reuses entries computed before the current window was filled, the same approximation Fast-dLLM (Wu et al. 2025) makes in its prefix cache mode. Fill-gating bounds the staleness, as the frozen prefix is one phase behind rather than an arbitrary number of steps, and Table 4 prices what it costs: 2.4 accuracy points averaged over the twelve settings, at most 0.6 on LLaDA Base and at worst 6.6 on Dream Instruct.

### F.2 Applicability to other model sizes and datasets

To fully validate our claims, we also incorporate ZENO(NB) sampler with Google's DiffusionGemma (Team et al. 2026) a 26B-A4B MoE diffusion model on a total of 8 benchmark datasets, including the six used in the main experiments. The two out of the distribution datasets (Namely, Math-500 (Hendrycks et al. 2021b) and GSM-Plus (Li et al. 2024)) are taken to showcase the plug-and-play nature of our ZENO sampler without any parameter tuning for either model or dataset.

| Dataset | Vanilla-GemmaSampler Acc (%) | NFE↓ | s/q↓ | Tok/s↑ | ZENO (NB) Acc (%) | NFE↓ | s/q↓ | Tok/s↑ | Gains ΔAcc | Speed-up |
|---|---|---|---|---|---|---|---|---|---|---|
| GSM8K | 95.0 | 11.3 | 1.2 | 232.2 | 95.5 | 13.1 | 1.2 | 236.8 | +0.5 | 1.0× |
| GSM-Plus | 77.2 | 14.6 | 1.5 | 223.6 | 78.6 | 16.1 | 1.5 | 214.2 | +1.4 | 1.0× |
| SVAMP | 93.0 | 9.7 | 0.9 | 196.6 | 93.0 | 11.2 | 0.9 | 210.9 | 0.0 | 1.1× |
| MATH-500 | 52.5 | 20.4 | 1.8 | 233.7 | 52.8 | 20.5 | 1.8 | 234.2 | +0.3 | 1.0× |
| MMLU | 79.5 | 22.0 | 2.1 | 163.6 | 79.2 | 23.7 | 2.1 | 160.1 | -0.3 | 1.0× |
| MMLU-Pro | 55.2 | 25.6 | 2.4 | 168.4 | 55.0 | 28.3 | 1.7 | 236.9 | -0.2 | 1.4× |
| HumanEval | 92.1 | 8.4 | 0.8 | 266.7 | 93.3 | 8.5 | 0.4 | 495.5 | +1.2 | 1.9× |
| MBPP | 78.8 | 9.9 | 0.9 | 140.5 | 77.3 | 10.5 | 0.5 | 254.8 | -1.5 | 1.8× |
| **Mean** | **77.9** | **15.2** | **1.4** | **203.2** | **78.1** | **16.5** | **1.3** | **255.4** | **+0.2** | **1.3**× |

Table 5: Per-dataset accuracy and decoding cost, Vanilla-GemmaSampler vs. ZENO(NB). ΔAcc = ZENO(NB) accuracy minus Vanilla accuracy. Speed-up is Vanilla's s/q divided by ZENO(NB)'s s/q, on the same checkpoint. The Mean row averages each column across the eight datasets.

We use the exact same set of parameter set for this experiment. The inference is done on $1\times$ NVIDIA-H200 GPU with a batch size of 1 and generation length ($L$) 512 and maximum step budget ($S$) as 512. From Table 5 it is evident that ZENO(NB) sampler outperforms the vanilla Gemma Sampler by significantly reducing the s/q latency by $1.3\times$. ZENO also pushes the average Tok/s throughput from $203.2 \rightarrow 255.4$ a jump of $52.2$ tokens per second. Finally, with an average accuracy gain of $0.2$ points, we establish ZENO as a model and dataset agnostic sampler.

### F.3 Generation Length and Step Budget

In this experiment, we vary both token budget or generation length $L$ and the step budget $S$ for LLaDA and Dream Instruct variants with 0-shot prompts. We compare three combinations of $(L/S)$: $256/256$, $512/256$ and $512/512$. From our experiment, it is evident that ZENO and all its variants are robust to generation length and step budget. One interesting finding is for the Vanilla, Block and SlowFast decoding (our closest competitor) baselines, the performance is really unstable and often degrades. But for ZENO as the step generation window and step budget increase, the accuracy actually improves within the similar fraction of NFE. This proves the robustness of ZENO in variable generation length and step budget setup. Detailed results are provided in Table 6

| Model | Method | GSM8K | | SVAMP | | MMLU | | MMLU-Pro | | MBPP | | HumanEval | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ |
| | **tokens/steps = 256/256** | | | | | | | | | | | | | | |
| LLaDA | Vanilla | 54.8 | 256 | 45.0 | 256 | 57.0 | 256 | **38.3**$^{tie}$ | 256 | 30.5 | 256 | 26.2 | 256 | 42.0 | 256 |
| | Block | 27.5 | 256 | 57.3 | 256 | 51.5 | 256 | 27.9 | 256 | 28.5 | 256 | 17.7 | 256 | 35.1 | 256 |
| | SlowFast | 71.5 | 123.1 | 84.3$^{tie}$ | 126.6 | **57.2** | 150.2 | **38.3**$^{tie}$ | 154.2 | 36.2 | 110.9 | **34.8** | 70.5 | 53.7 | 122.6 |
| | **ZENO (N)** | 72.3 | 96.0 | 78.0 | 93.5 | 50.8 | 107.5 | 35.0 | 111.0 | 38.0 | 93.3 | 28.7 | 77.0 | 50.5 | 96.4 |
| | **ZENO (NB)** | 74.8 | 107.9 | **87.7** | 104.4 | 55.8 | 133.7 | 34.8 | 137.2 | **38.3** | 96.7 | 34.2 | 88.2 | **54.2** | 111.4 |
| | **ZENO (NB) +C** | **75.3** | **56.9** | 84.3$^{tie}$ | **50.3** | 56.5 | **83.3** | 34.3 | **80.6** | 36.0 | **32.2** | 33.5 | **29.9** | 53.3 | **55.5** |
| | **tokens/steps = 512/256** | | | | | | | | | | | | | | |
| | Vanilla | 59.0 | 256 | 36.0 | 256 | **52.0** | 256 | **38.3** | 256 | 28.0 | 256 | 31.7 | 256 | 40.8 | 256 |
| | Block | 27.8 | 256 | 55.7 | 256 | 34.0 | 256 | 22.6 | 256 | 23.5 | 256 | 17.1 | 256 | 30.1 | 256 |
| | SlowFast | 79.2 | 140 | 86.0 | 116 | 50.3 | 161 | 30.9 | 183 | 33.5 | 82 | 39.6 | 87 | 53.3 | 128 |
| | **ZENO (N)** | 32.5 | 108 | 65.7 | 106 | 37.0 | **121** | 23.3 | **122** | 35.8 | 103 | 27.4 | 89 | 37.0 | 108 |
| | **ZENO (NB)** | **81.0** | 153 | 88.3 | 141 | 47.5 | 174 | 29.0 | 180 | **36.2** | 130 | **44.5** | 132 | **54.4** | 152 |
| | **ZENO (NB) +C** | 80.0 | **94** | **88.7** | **72** | 49.7 | 125 | 31.2 | 140 | 34.0 | **52** | 42.1 | **61** | 54.3 | **91** |
| | **tokens/steps = 512/512** | | | | | | | | | | | | | | |
| | Vanilla | 54.0 | 512 | 26.3 | 512 | 53.2 | 512 | **34.3** | 512 | 28.0 | 512 | 31.7 | 512 | 37.9 | 512 |
| | Block | 27.3 | 512 | 57.0 | 512 | 33.5 | 512 | 20.2 | 512 | 28.5 | 512 | 17.1 | 512 | 30.6 | 512 |
| | SlowFast | 79.2 | 140 | 86.0 | 116 | 50.3 | 161 | 30.9 | 183 | 33.5 | 82 | 39.6 | 87 | 53.3 | 128 |
| | **ZENO (N)** | 72.0 | 147 | 84.7 | 142 | 44.5 | 173 | 26.2 | 175 | **36.5**$^{tie}$ | 131 | 38.4 | 126 | 50.4 | 149 |
| | **ZENO (NB)** | **82.2** | 156 | 88.3 | 143 | 54.0 | 194 | 31.4 | 205 | **36.5**$^{tie}$ | 131 | **43.3** | 135 | **56.0** | 161 |
| | **ZENO (NB) +C** | 80.5 | **95** | **88.7** | **73** | **54.3** | **137** | 32.9 | **155** | 34.0 | **52** | 42.1 | **64** | 55.4 | **96** |
| | **tokens/steps = 256/256** | | | | | | | | | | | | | | |
| Dream | Vanilla | 40.8 | 256 | 76.7 | 256 | 61.3 | 256 | 40.0 | 256 | 44.8 | 256 | 29.3 | 256 | 48.8 | 256 |
| | Block | 13.5 | 256 | 38.3 | 256 | 41.5 | 256 | 21.2 | 256 | 28.5 | 256 | 20.1 | 256 | 27.2 | 256 |
| | SlowFast | 56.0 | 174.9 | 65.7 | 118.0 | **63.8** | 176.6 | **42.6** | 142.6 | 39.5 | 89.7 | 29.9 | 79.3 | 49.6 | 130.2 |
| | **ZENO (N)** | 52.5 | 97.6 | 77.7 | 94.5 | 33.0 | 107.3 | 22.6 | 109.3 | **54.2** | 82.7 | 50.0 | 77.7 | 48.3 | 94.9 |
| | **ZENO (NB)** | **79.8** | 110.4 | **87.7** | 102.7 | 56.2 | 136.7 | 37.6 | 136.7 | 54.0 | 85.1 | **54.3** | 83.5 | **61.6** | 109.2 |
| | **ZENO (NB) +C** | 76.3 | **63.0** | 86.3 | **49.5** | 58.0 | **99.4** | 38.6 | **100.5** | 53.3 | **29.7** | 48.8 | **29.3** | 60.2 | **61.9** |
| | **tokens/steps = 512/256** | | | | | | | | | | | | | | |
| | Vanilla | 35.8 | 256 | 70.3 | 256 | 53.5 | 256 | 32.6 | 256 | 41.0 | 256 | 21.3 | 256 | 42.4 | 256 |
| | Block | 11.8 | 256 | 29.7 | 256 | 37.7 | 256 | 21.7 | 256 | 26.2 | 256 | 13.4 | 256 | 23.4 | 256 |
| | SlowFast | 52.5 | 170 | 62.7 | 122 | **55.0** | 241 | **36.4** | 248 | 37.2 | 87 | 25.0 | 89 | 44.8 | 159 |
| | **ZENO (N)** | 40.2 | 114 | 72.7 | 111 | 16.3 | **124** | 7.1 | **126** | **55.2**$^{tie}$ | 100 | 45.1 | 97 | 39.5 | 112 |
| | **ZENO (NB)** | **77.8** | 156 | **86.7** | 144 | 34.0 | 193 | 22.9 | 195 | **55.2**$^{tie}$ | 126 | **53.0** | 125 | 54.9 | 157 |
| | **ZENO (NB) +C** | 75.8 | **91** | 85.3 | **72** | 41.5 | 172 | 30.0 | 178 | 54.2 | **42** | 49.4 | **46** | **56.0** | **100** |
| | **tokens/steps = 512/512** | | | | | | | | | | | | | | |
| | Vanilla | 32.5 | 512 | 71.3 | 512 | 56.8 | 512 | 35.9 | 512 | 37.2 | 512 | 21.9 | 512 | 42.6 | 512 |
| | Block | 13.8 | 512 | 39.0 | 512 | 36.2 | 512 | 21.9 | 512 | 28.2 | 512 | 21.3 | 512 | 26.7 | 512 |
| | SlowFast | 52.5 | 170 | 62.7 | 122 | 55.0 | 241 | 36.4 | 248 | 37.2 | 87 | 25.0 | 89 | 44.8 | 159 |
| | **ZENO (N)** | 62.0 | 157 | 82.0 | 145 | 24.3 | **186** | 13.6 | **187** | **55.5** | 126 | 47.6 | 128 | 47.5 | 155 |
| | **ZENO (NB)** | **80.5** | 156 | **88.7** | 145 | 60.2 | 238 | 39.1 | 244 | 55.2 | 126 | **51.8** | 127 | **62.6** | 173 |
| | **ZENO (NB) +C** | 78.0 | **92** | 86.7 | **71** | **61.2** | 198 | **39.5** | 210 | 54.2 | **42** | 48.8 | **47** | 61.4 | **110** |

Table 6: Accuracy (%) and NFE, grouped by model and segregated by *tokens*/*steps* setting within each model. **Bold** denotes the best result and underline the second-best result.

| Model | Method | GSM8K | | SVAMP | | MMLU | | MMLU-Pro | | MBPP | | HumanEval | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ |
| LLaDA | Vanilla | 63.0 | 256.0 | 71.3 | 256.0 | 56.5 | 256.0 | **40.2** | 256.0 | 30.8 | 256.0 | 20.1 | 256.0 | 47.0 | 256.0 |
| | Block | 43.2 | 256.0 | 76.3 | 256.0 | 56.3 | 256.0 | 34.1 | 256.0 | 19.2 | 256.0 | 18.9 | 256.0 | 41.4 | 256.0 |
| | ZENO (N) | 72.8 | 79.6 | 82.3 | 74.4 | **59.3** | 85.6 | 34.8 | 89.7 | 41.2 | 66.5 | 32.3 | 64.5 | 53.8 | 76.7 |
| | ZENO (NB) | **77.2** | 84.6 | 86.7 | 77.6 | 59.2 | 94.5 | 36.9 | 102.8 | **41.5** | 65.3 | **36.6** | 66.8 | **56.3** | 81.9 |
| | ZENO (NB) +C | 76.0 | **60.2** | **87.0** | **47.8** | 58.8 | **65.7** | 35.9 | **79.1** | 41.0 | **25.6** | 34.8 | **30.0** | 55.6 | **51.4** |
| Dream | Vanilla | 65.2 | 256.0 | 86.0 | 256.0 | 55.0 | 256.0 | 26.2 | 256.0 | 50.2 | 256.0 | 33.5 | 256.0 | 52.7 | 256.0 |
| | Block | 47.0 | 256.0 | 57.7 | 256.0 | 34.2 | 256.0 | 29.0 | 256.0 | 38.8 | 256.0 | 31.7 | 256.0 | 39.7 | 256.0 |
| | ZENO (N) | 79.8 | 72.3$^{\text{tie}}$ | 85.7 | 64.7 | 64.8 | 72.9 | 43.8 | 76.5 | 56.8 | 63.5 | **54.3**$^{\text{tie}}$ | 64.5 | 64.2 | 69.1 |
| | ZENO (NB) | 81.5 | 72.3$^{\text{tie}}$ | **86.7** | 64.4 | **66.8** | 73.5 | **45.0** | 78.4 | **57.0** | 61.7 | **54.3**$^{\text{tie}}$ | 63.3 | **65.2** | 68.9 |
| | ZENO (NB) +C | **82.2** | **35.4** | 85.7 | **22.9** | 66.3 | **33.7** | 44.0 | **42.1** | 55.8 | **20.8** | 51.8 | **24.3** | 64.3 | **29.9** |

Table 7: Instruction-tuned checkpoints evaluated with few-shot prompting examples. **Bold** denotes the best result and underline the second-best result.

| Method | LLaDA | | | | | | Dream | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc (%) | NFE↓ | s/q↓ | Tok/s↑ | Speed-up$_{s/q}$ | Speed-up$_{Tok/s}$ | Acc (%) | NFE↓ | s/q↓ | Tok/s↑ | Speed-up$_{s/q}$ | Speed-up$_{Tok/s}$ |
| Vanilla | 47.0 | 256.0 | 25.8 | 10.6 | 1.00× | 1.00× | 52.7 | 256.0 | 22.2 | 12.4 | 1.00× | 1.00× |
| Block | 41.4 | 256.0 | 39.1 | 7.5 | 0.66× | 0.70× | 39.7 | 256.0 | 23.1 | 12.0 | 0.96× | 0.97× |
| ZENO (N) | 53.8 | 76.7 | 8.0 | 36.1 | 3.22× | 3.40× | 64.2 | 69.1 | 6.2$^{\text{tie}}$ | 46.4 | 3.60×$^{\text{tie}}$ | 3.74× |
| ZENO (NB) | **56.3** | 81.9 | 8.6 | 34.3 | 3.00× | 3.24× | **65.2** | 68.9 | 6.2$^{\text{tie}}$ | 46.6 | 3.60×$^{\text{tie}}$ | 3.76× |
| ZENO (NB) +C | 55.6 | **51.4** | **5.7** | **61.3** | **4.55**× | **5.78**× | 64.3 | **29.9** | **2.8** | **114.0** | **7.86**× | **9.20**× |

Table 8: Accuracy and decoding cost for (Table 7), averaged over the same six benchmarks.

### F.4 Few-Shot Prompting on Instruction-Tuned Checkpoints

Results for Instruction-tuned checkpoints in the main section (Table 1) are mainly for the zero-shot prompting setup. To show the robustness in the few-shot prompting setup we provide the few-shot prompt ablation for the instruction-tuned checkpoints. As we already establish the superior performance of ZENO sampler, to save evaluation costs and to show a fair comparison, we restrict this ablation to Vanilla, Block and the three ZENO variants. We use the exact same few-shot prompt used in Base model experiments for this ablation. Our experiments show that few-shot prompting improves the average accuracy across all the baselines, but the difference between Vanilla is still over 9.3 and 12.5 points for LLaDA and Dream, respectively. An interesting finding from this ablation setup is that ZENO in few-shot prompting setting has much higher throughput than the vanilla and Block sampler, with fewer number of steps than zero-shot prompting evaluation setup. This indicates towards ZENO's robustness and better efficiency in different prompting setups. One thing to note is that the individual benchmarking scores do not match the actual paper's numbers because the few-shot prompts used in their evaluation are not disclosed by the respective authors. For all our experiments, we use our own curated prompts and keep it uniform across all setups. The detailed results are provided in Table 7 and 8.

### F.5 Growth Base: How Fast Should the Window Widen?

The growth base $\gamma$ in Eq. 3 sets how aggressively the window expands between phases. Small $\gamma$ produces many narrow phases, keeping the sampler close to left-to-right order for longer; large $\gamma$ reaches the full canvas in a handful of phases, exposing distant positions early. Table 9 we ablate all three setups for ZENO with four different growth base: $\gamma \in \{2, 3, 4, 5\}$ with every other parameter fixed. The default $\gamma = 2$ is used throughout the paper. From the table it is also evident that $\gamma$ of 2 is a decent setup for stable output across benchmarks. The detailed results and numbers are provided in Table 9.

### F.6 Block Length After the Switch Width

Once the widening window reaches $W^\star$, ZENO (NB) and ZENO (NB) +C stop geometrically growing and advance in fixed-size blocks. To find the best setting for the block length, we perform the post-switch block length ablation experiment. Across four different block lengths, $\{16, 32, 64, 128\}$ for both ZENO (NB) and ZENO (NB) +C the experiment is performed. From the trend observed in the table it is evident that the throughput increases as the block length increases. But there is an accuracy-throughput tradeoff. Keeping the block length too big improves the efficiency but reduces

| Model | Method | growth_base | GSM8K | SVAMP | MMLU | MMLU-Pro | MBPP | HumanEval | Mean |
|---|---|---|---|---|---|---|---|---|---|
| LLaDA | ZENO (N) | 2 | 72.2 | 78.0 | 50.8$^{tie}$ | 35.0 | 38.0 | 28.7 | 50.5 |
| | | 3 | 72.2 | 78.3 | **51.8** | 36.2 | 37.2 | 32.3 | 51.4 |
| | | 4 | 65.8 | 54.0 | 49.0 | **36.7** | 37.2 | 29.9 | 45.4 |
| | | 5 | **75.5** | **83.3** | 50.8$^{tie}$ | 36.4 | **39.0** | **32.9** | **53.0** |
| | ZENO (NB) | 2 | 74.8 | **87.7** | 54.3 | 34.8 | 38.2 | 34.1 | 54.0 |
| | | 3 | 74.5 | 86.0 | **54.7** | 36.4 | 39.0 | **36.6** | **54.5** |
| | | 4 | 73.5 | 87.0 | 53.3 | 36.7 | **39.5** | 36.0$^{tie}$ | 54.0 |
| | | 5 | **75.0** | 86.7 | 52.0 | **37.6** | 38.5 | 36.0$^{tie}$ | 54.3 |
| | ZENO (NB) +C | 2 | 75.2 | 84.3 | **55.3** | 34.3 | 36.0 | 33.5 | 53.1 |
| | | 3 | 75.5 | **84.7**$^{tie}$ | 53.7 | 36.4 | 36.2$^{tie}$ | 32.9 | 53.2$^{tie}$ |
| | | 4 | 75.0 | 83.3 | 53.9 | 36.0 | **37.0** | **34.1**$^{tie}$ | 53.2$^{tie}$ |
| | | 5 | **76.2** | **84.7**$^{tie}$ | 51.6 | **36.9** | 36.2$^{tie}$ | **34.1**$^{tie}$ | **53.3** |
| Dream | ZENO (N) | 2 | 52.5 | 77.7 | 33.0 | 22.6 | 54.0 | **50.0** | 48.3 |
| | | 3 | 66.0 | **83.7** | 46.7 | **32.4** | **54.5** | 45.7 | **54.8** |
| | | 4 | 34.2 | 67.0 | 33.3 | 20.2 | 51.2 | 47.0 | 42.2 |
| | | 5 | **66.8** | 73.0 | **47.1** | 32.1 | 53.0 | 48.2 | 53.4 |
| | ZENO (NB) | 2 | **79.0** | **88.0** | 55.9 | 37.6 | **54.0** | **53.7** | 61.4 |
| | | 3 | 76.5 | 83.0 | **63.7** | 40.0 | 53.0 | 50.0 | 61.0 |
| | | 4 | 77.8 | 87.7 | 62.5 | **40.2** | 51.7 | 52.4 | **62.1** |
| | | 5 | 75.5 | 81.7 | 62.9 | 38.1 | 50.7 | 50.0 | 59.8 |
| | ZENO (NB) +C | 2 | **76.2**$^{tie}$ | **86.3**$^{tie}$ | 57.1 | 38.6 | **53.2** | 48.8 | 60.0 |
| | | 3 | 74.5$^{tie}$ | 82.0 | 62.9 | 39.8 | 51.2 | 47.0 | 59.6 |
| | | 4 | **76.2**$^{tie}$ | **86.3**$^{tie}$ | 62.2 | **40.2** | 50.7 | **50.6** | **61.1** |
| | | 5 | 74.5$^{tie}$ | 79.7 | **63.5** | 39.0 | 48.8 | 47.0 | 58.7 |

Table 9: Ablation over the `growth_base` parameter for **ZENO (N)**, **ZENO (NB)**, and **ZENO (NB) +C** on LLaDA-8B-Instruct and Dream-7B-Instruct. Accuracy (%) reported per dataset. This sweep is a separate run series from the main table, so the $\gamma = 2$ rows are close to, but not numerically identical with, the corresponding rows of Table 1. **Bold** = best, underline = second-best (closest to max), per column, within each method's four `growth_base` settings. Superscript $^{tie}$ marks values tied with another row in that column.

| Model | Method | block_size | GSM8K | | SVAMP | | MMLU | | MMLU-Pro | | MBPP | | HumanEval | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Acc | Tok/s↑ | Acc | Tok/s↑ | Acc | Tok/s↑ | Acc | Tok/s↑ | Acc | Tok/s↑ | Acc | Tok/s↑ | Acc | TPS↑ |
| LLaDA | ZENO (NB) | 16 | **75.2**$^{tie}$ | 68.0 | 87.3 | 61.7 | 52.8 | 30.6 | 29.3 | 27.5 | 38.5$^{tie}$ | 40.5 | **36.6** | 54.2 | 53.3$^{tie}$ | 47.0 |
| | | 32 | 74.8 | 70.2 | 87.0 | 63.6 | **55.2** | 31.3 | 28.8 | 28.3 | 38.5$^{tie}$ | 42.7 | 35.4 | 56.6 | 53.3$^{tie}$ | 48.8 |
| | | 64 | **75.2**$^{tie}$ | 71.5 | **87.7** | **65.0** | 54.5 | 32.2 | 29.5 | 29.6 | **38.8** | 43.8 | 36.0 | 57.8 | **53.6** | 50.0 |
| | | 128 | 74.0 | **73.0** | 83.0 | 64.7 | 52.2 | **35.4** | **30.5** | **32.5** | 38.5$^{tie}$ | **44.1** | 28.7 | **61.7** | 51.1 | **51.9** |
| | ZENO (NB) +C | 16 | **75.5** | 92.9 | **86.7** | 88.9 | 53.0 | 37.8 | 29.5$^{tie}$ | 35.3 | **36.5**$^{tie}$ | 75.6 | **35.4** | 104.0 | **52.8** | 72.4 |
| | | 32 | 73.0 | 98.8 | 85.0 | 95.0 | **54.5** | 39.1 | 29.5$^{tie}$ | 36.8 | 36.2 | 84.3 | 32.9 | 118.2 | 51.9 | 78.7 |
| | | 64 | 75.0 | 103.1 | 85.3 | **98.4** | 54.0 | 40.2 | **30.5** | 38.3 | **36.5**$^{tie}$ | 89.0 | 34.8 | 123.9 | 52.7 | 82.2 |
| | | 128 | 72.0 | **104.6** | 83.7 | 97.2 | 53.7 | **43.9** | 29.5$^{tie}$ | **41.3** | 36.0 | **90.4** | 28.7 | **129.0** | 50.6 | **84.4** |
| Dream | ZENO (NB) | 16 | **81.8** | 46.0 | **87.3**$^{tie}$ | 37.5 | 59.5 | 34.6 | 35.7 | 32.8 | 54.0 | 17.9 | 51.2 | 30.0 | 61.6 | 33.1 |
| | | 32 | 81.2 | **47.3** | **87.3**$^{tie}$ | 39.5 | **60.8** | 34.8 | **38.3** | 33.0 | **54.2**$^{tie}$ | 18.8 | **51.8**$^{tie}$ | 31.7 | **62.3** | 34.2 |
| | | 64 | 80.0 | 47.0 | **87.3**$^{tie}$ | **40.1** | 52.2 | 35.0 | 32.9 | 33.3 | **54.2**$^{tie}$ | **19.1** | **51.8**$^{tie}$ | **32.3** | 59.7 | 34.5 |
| | | 128 | 63.8 | 46.8 | 83.0 | 39.7 | 34.7 | **37.8** | 20.2 | **35.8** | **54.2**$^{tie}$ | 18.6 | 49.4 | 31.8 | 50.9 | **35.1** |
| | ZENO (NB) +C | 16 | **80.2** | 62.5 | 86.0 | 56.6 | 60.0 | 40.2$^{tie}$ | 34.3 | 37.8 | 53.5 | 35.2 | **50.0** | 58.9 | 60.7 | 48.5 |
| | | 32 | 78.5 | 65.8 | 85.7 | 60.8 | **61.5** | 40.2$^{tie}$ | **37.6** | 38.1 | **53.8**$^{tie}$ | 38.9 | 48.8$^{tie}$ | 66.9 | **61.0** | 51.8 |
| | | 64 | 79.5 | **66.0** | **86.3** | **62.7** | 54.5 | 40.2$^{tie}$ | 36.7 | 38.3 | **53.8**$^{tie}$ | **41.0** | 48.8$^{tie}$ | **70.6** | 59.9 | **53.1** |
| | | 128 | 68.5 | 62.8 | 82.7 | 61.6 | 38.8 | **42.4** | 25.2 | **39.9** | **53.8**$^{tie}$ | 40.2 | 48.8$^{tie}$ | 66.7 | 53.0 | 52.3 |

Table 10: Accuracy (%) and **true** tokens/second as the post-switch block length of **ZENO (NB)** and **ZENO (NB) +C** varies. Batch size 1, so every question is timed exactly.

accuracy. Across all the block lengths, a block of 64 performs the best in terms of accuracy and throughput balance. Detailed results are shown in Table 10.

### F.7 Confidence Threshold Ablation

To observe the effect of confidence threshold on generation speed and quality, we perform the ablation with three distinct confidence values $\tau \in \{0.6, 0.8, 0.9\}$. The other parameters like switch width, generation window, and temperature we keep fixed as the main experiment. This experiment reveals that a $\tau$ of 0.8 sits at the optimal point of maximum accuracy-efficiency tradeoff. Table 11 contains the detailed results.

| Model | Method | Confidence Threshold | GSM8K | | SVAMP | | MMLU | | MMLU-Pro | | MBPP | | HumanEval | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ | Acc | NFE↓ |
| LLaDA Instruct | ZENO (NB) +C | 0.6 | 71.0 | **37** | 82.7 | **32** | 51.0 | **61** | 29.5 | **60** | 29.8 | **25** | 25.6 | **22** | 48.3 | **39** |
| | | 0.8 | **75.3** | 57 | 84.3 | 50 | **56.5** | 83 | **34.3** | 81 | 36.0 | 32 | **33.5** | 30 | **53.3** | 56 |
| | | 0.9 | 75.0 | 74 | **86.7** | 66 | 54.0 | 95 | 29.3 | 93 | **39.0** | 39 | 32.9 | 38 | 52.8 | 67 |
| Dream Instruct | ZENO (NB) +C | 0.6 | 65.5 | **42** | 75.3 | **34** | 57.5 | **73** | 35.5 | **74** | 44.2 | **23** | 30.5 | **22** | 51.4 | **45** |
| | | 0.8 | 76.3 | 63 | 86.3 | 50 | **58.0** | 99 | **38.6** | 101 | 53.3 | 30 | 48.8 | 29 | **60.2** | 62 |
| | | 0.9 | **80.2** | 80 | **87.3** | 61 | 44.8 | 113 | 29.0 | 114 | **55.2** | 36 | **53.7** | 36 | 58.4 | 73 |
| LLaDA Base | ZENO (NB) +C | 0.6 | 69.8 | **58** | 80.3 | **42** | 57.0 | **56** | **37.9** | **75** | 37.8 | **49** | 25.6 | **31** | 51.4 | **52** |
| | | 0.8 | **77.2**$^{\text{tie}}$ | 76 | **83.3** | 58 | 56.2 | 67 | 37.6$^{\text{tie}}$ | 93 | **40.8**$^{\text{tie}}$ | 70 | **30.5**$^{\text{tie}}$ | 41 | **54.3**$^{\text{tie}}$ | 68 |
| | | 0.9 | **77.2**$^{\text{tie}}$ | 90 | 82.0 | 66 | **57.5** | 75 | 37.6$^{\text{tie}}$ | 101 | **40.8**$^{\text{tie}}$ | 87 | **30.5**$^{\text{tie}}$ | 50 | **54.3**$^{\text{tie}}$ | 78 |
| Dream Base | ZENO (NB) +C | 0.6 | 66.5 | **80** | 77.3 | **54** | 63.8 | **86** | 43.3 | **83** | 46.0 | **61** | 41.5 | **60** | 56.4 | **71** |
| | | 0.8 | **74.0** | 104 | **79.3**$^{\text{tie}}$ | 62 | **65.8** | 98 | **44.5** | 99 | 53.5 | 82 | 53.0 | 77 | **61.7**$^{\text{tie}}$ | 87 |
| | | 0.9 | 73.0 | 116 | **79.3**$^{\text{tie}}$ | 74 | 65.3 | 106 | 44.3 | 108 | **54.8** | 97 | **53.7** | 94 | **61.7**$^{\text{tie}}$ | 99 |

Table 11: Accuracy (%) and **measured** forward evaluations per question as the confidence threshold of **ZENO (NB) +C** varies, on instruction-tuned checkpoints (zero-shot) and base checkpoints (few-shot exemplars supplied as chat turns), on the full evaluation sets. All other settings are held at the main-sweep configuration: 256 steps, growth base 2, initial width 4, switch width 128, constant block length 64, greedy decoding. **Bold** = best, underline = second-best

### F.8 Sampled Decoding at Temperature 0.7: Per-Seed Results

To verify the robustness of our scheduler, we test ZENO in non-deterministic setup. For this we employ a decoding temperature $\tau_{\text{gen}} = 0.7$ and running five independent seeds (42, 123, 512, 1024, 4096). Table 13 shows the comparison between vanilla vs all three of the ZENO sampler variants for both models. Our experiments show stochastic sampling moves the means very little. Every ZENO variant is within $1-2$ points of its greedy accuracy at the same budget (Table 6). In this setup as well ZENO (NB) outperforms the vanilla decoder by over 11.4 and 9.4 points, which closely matches the deterministic performance of the samplers. This establishes the robustness of the ZENO class of sampler in diverse settings.

## G Computational Statistics

All experiments are run on NVIDIA A100-SXM4-80GB GPUs. Because ZENO is a test-time scaling method, there is no associated cost of training. Table 12 breaks down the measured decoding time across the sweeps reported in this paper. A total of approximately 1,955.9 hours of GPU compute time was required for all the experiments across the paper, excluding the unit tests. Among all the experiments, the length/step sweep is the most expensive block, taking around 474.8 GPU-hours. The reported time excludes model loading, tokenization, answer extraction and validation delays. The details are provided in Table 12.

| Sweep | Decode time (GPU-h) |
|---|---|
| Main comparison, instruction-tuned models | 434.2 |
| Main comparison, base models | 318.8 |
| Few-shot prompting, instruction-tuned models | 191.5 |
| Generation length × step budget | 474.8 |
| Sampled decoding, 5 seeds at $\tau_{\text{gen}} = 0.7$ | 321.7 |
| ZENO schedule ablations (window growth, block size) | 95.9 |
| Confidence threshold, instruction-tuned | 22.7 |
| Confidence threshold, base | 65.2 |
| KV-cache study | 31.1 |
| **Total** | **1,955.9** |

Table 12: Measured decode time for the nine sweeps listed, summed over the evaluation cells of each. Figures are inference only – ZENO trains nothing – and cover the sampler call alone, excluding model loading, tokenization and answer extraction.

| Model | Method | Seed | GSM8K | SVAMP | MMLU | MMLU-Pro | MBPP | HumanEval | Mean |
|---|---|---|---|---|---|---|---|---|---|
| LLaDA | Vanilla | 42 | 57.0 | 52.0 | 52.8 | 30.2 | 31.8 | 24.4 | 41.4 |
| | | 123 | 57.8 | 46.3 | 51.0 | 33.1 | 29.5 | 26.2 | 40.7 |
| | | 512 | 58.5 | 49.7 | 52.2 | 32.6 | 29.5 | 28.7 | 41.9 |
| | | 1024 | 57.5 | 49.0 | 51.8 | 33.1 | 30.8 | 28.1 | 41.7 |
| | | 4096 | 58.2 | 48.3 | 52.5 | 33.1 | 31.8 | 28.1 | 42.0 |
| | | *mean* | 57.8 ±0.6 | 49.1 ±2.1 | 52.1 ±0.7 | **32.4** ±1.2 | 30.6 ±1.1 | 27.1 ±1.8 | 41.5 ±0.5 |
| | **ZENO (N)** | 42 | 70.8 | 76.0 | 48.5 | 32.6 | 36.8 | 26.8 | 48.6 |
| | | 123 | 72.2 | 78.0 | 46.0 | 28.8 | 36.8 | 27.4 | 48.2 |
| | | 512 | 68.8 | 76.7 | 47.7 | 30.0 | 36.2 | 28.7 | 48.0 |
| | | 1024 | 71.8 | 74.3 | 47.0 | 31.2 | 36.5 | 27.4 | 48.0 |
| | | 4096 | 67.5 | 73.0 | 49.2 | 31.9 | 34.5 | 29.3 | 47.6 |
| | | *mean* | 70.2 ±2.0 | 75.6 ±2.0 | 47.7 ±1.2 | 30.9 ±1.5 | 36.1 ±0.9 | 27.9 ±1.0 | 48.1 ±0.4 |
| | **ZENO (NB)** | 42 | 76.2 | 87.3 | 51.7 | 30.0 | 36.2 | 37.2 | 53.1 |
| | | 123 | 75.0 | 87.7 | 49.3 | 28.3 | 38.0 | 38.4 | 52.8 |
| | | 512 | 75.2 | 86.3 | 53.2 | 28.6 | 38.5 | 37.2 | 53.2 |
| | | 1024 | 75.5 | 87.3 | 53.3 | 28.8 | 37.5 | 32.9 | 52.6 |
| | | 4096 | 77.5 | 86.7 | 52.3 | 28.8 | 37.0 | 36.0 | 53.1 |
| | | *mean* | **75.9** ±1.0 | **87.1** ±0.5 | 52.0 ±1.6 | 28.9 ±0.6 | **37.5** ±0.9 | **36.3** ±2.1 | **52.9** ±0.3 |
| | **ZENO (NB) +C** | 42 | 71.8 | 84.7 | 51.8 | 30.7 | 36.2 | 32.9 | 51.4 |
| | | 123 | 74.0 | 86.0 | 54.2 | 33.1 | 36.8 | 32.9 | 52.8 |
| | | 512 | 75.8 | 87.0 | 54.2 | 28.3 | 35.0 | 31.1 | 51.9 |
| | | 1024 | 76.2 | 83.7 | 50.2 | 31.7 | 35.0 | 34.2 | 51.8 |
| | | 4096 | 73.8 | 86.7 | 51.2 | 28.6 | 37.2 | 34.2 | 52.0 |
| | | *mean* | 74.3 ±1.8 | 85.6 ±1.4 | **52.3** ±1.8 | 30.5 ±2.0 | 36.0 ±1.0 | 33.1 ±1.3 | 52.0 ±0.5 |
| Dream | Vanilla | 42 | 46.2 | 76.0 | 60.2 | 39.3 | 49.0 | 30.5 | 50.2 |
| | | 123 | 47.0 | 79.3 | 58.2 | 36.2 | 46.8 | 34.8 | 50.4 |
| | | 512 | 49.8 | 77.7 | 56.5 | 35.7 | 48.0 | 36.6 | 50.7 |
| | | 1024 | 51.2 | 77.7 | 58.8 | 35.5 | 49.5 | 34.2 | 51.2 |
| | | 4096 | 49.5 | 77.7 | 62.3 | 37.9 | 48.5 | 32.9 | 51.5 |
| | | *mean* | 48.8 ±2.1 | 77.7 ±1.2 | **59.2** ±2.2 | **36.9** ±1.6 | 48.4 ±1.1 | 33.8 ±2.3 | 50.8 ±0.5 |
| | **ZENO (N)** | 42 | 52.5 | 75.7 | 26.5 | 14.5 | 52.5 | 48.8 | 45.1 |
| | | 123 | 48.8 | 76.0 | 28.2 | 14.8 | 49.8 | 46.3 | 44.0 |
| | | 512 | 50.0 | 75.7 | 30.7 | 16.9 | 56.0 | 47.6 | 46.2 |
| | | 1024 | 53.2 | 75.7 | 26.0 | 16.4 | 53.5 | 47.6 | 45.4 |
| | | 4096 | 52.8 | 74.0 | 27.7 | 17.6 | 53.8 | 46.3 | 45.4 |
| | | *mean* | 51.5 ±2.0 | 75.4 ±0.8 | 27.8 ±1.8 | 16.0 ±1.4 | 53.1 ±2.3 | 47.3 ±1.0 | 45.2 ±0.8 |
| | **ZENO (NB)** | 42 | 81.0 | 87.0 | 53.3 | 32.9 | 53.2 | 52.4 | 60.0 |
| | | 123 | 80.0 | 86.3 | 53.0 | 36.9 | 52.2 | 53.0 | 60.2 |
| | | 512 | 78.2 | 87.0 | 53.8 | 32.4 | 53.0 | 51.2 | 59.3 |
| | | 1024 | 80.2 | 85.3 | 55.5 | 35.9 | 54.8 | 56.1 | 61.3 |
| | | 4096 | 80.5 | 86.7 | 55.2 | 33.8 | 53.5 | 53.0 | 60.5 |
| | | *mean* | **80.0** ±1.0 | **86.5** ±0.7 | 54.2 ±1.1 | 34.4 ±2.0 | **53.4** ±0.9 | **53.2** ±1.8 | **60.2** ±0.7 |
| | **ZENO (NB) +C** | 42 | 79.2 | 83.7 | 55.3 | 36.9 | 49.8 | 51.2 | 59.4 |
| | | 123 | 78.2 | 86.0 | 54.3 | 36.4 | 51.7 | 48.2 | 59.1 |
| | | 512 | 79.5 | 86.0 | 59.2 | 35.7 | 53.0 | 48.8 | 60.4 |
| | | 1024 | 78.8 | 86.0 | 54.8 | 33.8 | 51.7 | 50.0 | 59.2 |
| | | 4096 | 76.8 | 85.0 | 56.3 | 34.1 | 54.0 | 48.8 | 59.2 |
| | | *mean* | 78.5 ±1.1 | 85.3 ±1.0 | 56.0 ±1.9 | 35.4 ±1.4 | 52.0 ±1.6 | 49.4 ±1.2 | 59.4 ±0.5 |

Table 13: Per-seed accuracy (%) at temperature 0.7. Each of the five seeds (42, 123, 512, 1024, 4096) is an independent sampled decode of the whole benchmark and is reported individually; the *mean* row gives the average over those five runs with the standard deviation. The *Mean* column averages across the six benchmarks. All settings use a 256-token budget and a 256-step schedule. Vanilla, `ZENO(N)` and `ZENO(NB)` are decoded at batch size 1; the `ZENO(NB)+C` rows come from a batch-1 run of the same configuration, which changes its six-benchmark mean by less than the seed-to-seed standard deviation. The table reports accuracy only, so no timing comparison with the rest of the paper is implied. Marks appear on the *mean* rows only, comparing methods within a model: **bold** = best, underline = second-best per column. Individual seed rows are left unmarked.

## H ALGORITHM

In Algorithm 1 we present the ZENO sampler formulation. It is a drop-in replacement for the sampling loop of a masked diffusion language model. It simply takes the same frozen network ($f_\theta$), the same fully-masked canvas, and the same nominal step budget and differs only in which masked positions are eligible for commitment at each step and how many of them are committed.

**Algorithm 1** ZENO sampler decoding for a sequence

**Require:** frozen model $f_\theta$; prompt of length $P_r$; generation length $L$; nominal step budget $S \geq 1$; mode $\in \{\text{QUOTA}, \text{THRESHOLD}\}$; schedule parameters $(w_0, \gamma, \kappa, W^\star, \beta)$; threshold $\tau$; integer reveal cap $M \geq 1$

$x \leftarrow [\,\text{prompt} \,\|\, \texttt{[MASK]}^{L}\,]$
$\mathcal{W} \leftarrow$ zone widths from Eq. 3 or Eq. 5
$K \leftarrow |\mathcal{W}|$; $S' \leftarrow \lceil S/K \rceil$; NFE $\leftarrow 0$
**for** $w \in \mathcal{W}$ **do**
  $\mathcal{E} \leftarrow \{p+\delta : 0 \leq \delta < w,\ x_{p+\delta} = \texttt{[MASK]}\}$
  $n \leftarrow |\mathcal{E}|$; $t \leftarrow 0$
  $B \leftarrow B(n, S')$ in threshold mode; otherwise $B \leftarrow S'$
  **if** mode $=$ QUOTA **then**
    $a \leftarrow \lfloor n/S' \rfloor$; $r \leftarrow n \bmod S'$
    $k_j \leftarrow a + \mathbf{1}[j \leq r]$, for $j = 1, \ldots, S'$ ▷ $\sum_j k_j = n$
  **end if**
  **while** $\mathcal{E} \neq \emptyset$ **and** $t < B$ **do**
    $t \leftarrow t + 1$
    **if** mode $=$ QUOTA and $k_t = 0$ **then**
      **continue** ▷ no forward pass for zero quota
    **end if**
    $\ell \leftarrow f_\theta(x)$; NFE $\leftarrow$ NFE $+ 1$
    $(\hat{x}, c) \leftarrow$ candidates and scores from Eqs. 1–2
    $\tilde{c}_i \leftarrow \begin{cases} c_i, & i \in \mathcal{E}, \\ -\infty, & i \notin \mathcal{E} \end{cases}$
    **if** mode $=$ QUOTA **then**
      $R \leftarrow \mathrm{TopK}(\mathcal{E}, \tilde{c}, k_t)$
    **else**
      $R \leftarrow \{i \in \mathcal{E} : \tilde{c}_i \geq \tau\}$
      **if** $|R| > M$ **then**
        $R \leftarrow \mathrm{TopK}(R, \tilde{c}, M)$
      **end if**
      **if** $R = \emptyset$ **then**
        $R \leftarrow \{\arg\max_{i \in \mathcal{E}} \tilde{c}_i\}$ ▷ forced progress
      **end if**
    **end if**
    $x_i \leftarrow \hat{x}_i$ for all $i \in R$
    $\mathcal{E} \leftarrow \mathcal{E} \setminus R$
  **end while**
**end for**
**if** mode $=$ THRESHOLD and $x$ contains [MASK] **then**
  $\ell \leftarrow f_\theta(x)$; NFE $\leftarrow$ NFE $+ 1$
  recompute $\hat{x}$ from the fresh logits and commit at every still-masked position ▷ final cleanup pass
**end if**
**return** $x$, NFE

**Evaluated configurations.** All reported ZENO variants use threshold mode with $\tau = 0.8$. ZENO(N) and ZENO(NB) use $M = 5$, whereas ZENO(NB)+C uses $M = 64$. In Algorithm 1, $B(n, S')$ counts the nonzero entries of the linear transfer schedule for $n$ masked positions and $S'$ nominal steps. Unresolved masks carry into the next zone; final cleanup bypasses $\tau$ and $M$.

# I OUTPUT ANALYSIS

**Paired Correctness and Error Analysis.** In this section, we provide the paired correctness with respect to the vanilla decoding setup using internal data statistics and detailed qualitative analysis in the following section. For each of the LLaDA and Dream-Base and Instruct series, we compute four numbers. Case where both solutions are correct; case where both solutions are wrong; case where ZENO gives the correct solution and finally, the case where vanilla decoding gives the correct solution. This analysis reveals that in cases like mathematical reasoning and coding, the directional decoding paradigm of ZENO helps in generating correct solutions by a solid margin. However, in memorization tasks ZENO falls a bit short in most of the cases.

| | | | LLaDA | | | | | Dream | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Regime** | **Method** | **Dataset** | Both✓ | Both× | Ours only✓ | Van. only✓ | Δ | Both✓ | Both× | Ours only✓ | Van. only✓ | Δ |
| **Instruct** | **Vanilla vs. ZENO (NB)** | GSM8K | 200 | 82 | 99 | 19 | +80 | 156 | 74 | 163 | 7 | **+156** |
| | | SVAMP | 129 | 31 | 134 | 6 | **+128** | 218 | 25 | 45 | 12 | +33 |
| | | MMLU | 282 | 205 | 53 | 60 | -7 | 269 | 164 | 68 | 99 | -31 |
| | | MMLU-Pro | 113 | 226 | 33 | 48 | -15 | 109 | 203 | 49 | 59 | -10 |
| | | MBPP | 96 | 221 | 57 | 26 | +31 | 157 | 162 | 59 | 22 | +37 |
| | | HumanEval | 35 | 100 | 21 | 8 | +13 | 44 | 71 | 45 | 4 | +41 |
| | **Vanilla vs. ZENO (NB) +C** | GSM8K | 202 | 82 | 99 | 17 | +82 | 153 | 85 | 152 | 10 | **+142** |
| | | SVAMP | 125 | 37 | 128 | 10 | **+118** | 216 | 27 | 43 | 14 | +29 |
| | | MMLU | 283 | 202 | 56 | 59 | -3 | 280 | 164 | 68 | 88 | -20 |
| | | MMLU-Pro | 112 | 227 | 32 | 49 | -17 | 109 | 199 | 53 | 59 | -6 |
| | | MBPP | 90 | 224 | 54 | 32 | +22 | 156 | 164 | 57 | 23 | +34 |
| | | HumanEval | 34 | 100 | 21 | 9 | +12 | 43 | 79 | 37 | 5 | +32 |
| **Base** | **Vanilla vs. ZENO (NB)** | GSM8K | 290 | 76 | 20 | 14 | +6 | 275 | 84 | 20 | 21 | -1 |
| | | SVAMP | 235 | 41 | 15 | 9 | +6 | 223 | 47 | 17 | 13 | +4 |
| | | MMLU | 298 | 224 | 42 | 36 | +6 | 357 | 174 | 39 | 30 | +9 |
| | | MMLU-Pro | 136 | 228 | 25 | 31 | -6 | 160 | 208 | 26 | 26 | 0 |
| | | MBPP | 149 | 218 | 14 | 19 | -5 | 211 | 163 | 8 | 18 | -10 |
| | | HumanEval | 48 | 110 | 3 | 3 | 0 | 78 | 70 | 8 | 8 | 0 |
| | **Vanilla vs. ZENO (NB) +C** | GSM8K | 290 | 77 | 19 | 14 | +5 | 276 | 84 | 20 | 20 | 0 |
| | | SVAMP | 235 | 41 | 15 | 9 | +6 | 221 | 47 | 17 | 15 | +2 |
| | | MMLU | 298 | 227 | 39 | 36 | +3 | 357 | 175 | 38 | 30 | +8 |
| | | MMLU-Pro | 134 | 229 | 24 | 33 | -9 | 160 | 207 | 27 | 26 | +1 |
| | | MBPP | 149 | 218 | 14 | 19 | -5 | 207 | 164 | 7 | 22 | -15 |
| | | HumanEval | 48 | 111 | 2 | 3 | -1 | 77 | 68 | 10 | 9 | +1 |

Table 14: Per-question paired correctness breakdown between **ZENO (NB)** / **ZENO (NB) +C** and **Vanilla**, on Instruct and Base models, at temperature 0 (greedy) seed 42. **Both✓** = both correct; **Both×** = both wrong; **Ours only✓** = our method correct, Vanilla wrong; **Van. only✓** = Vanilla correct, ours wrong; Δ = Ours only − Van. only, the net swing in our method's favor.

To investigate further, we look into the generated trajectory and find that, despite being directed to do reasoning, the vanilla sampler often in many cases generates the final answer directly, ignoring the reasoning instruction in both base and instruction-tuned cases. Also, oftentimes for ZENO the generation window is exhausted while reasoning. Table 6 showcases when generation length/decoding step is set to 512/512, ZENO closely matches and in many cases, outperforms the vanilla decoding setup. With this ZENO also maintains its high performance on reasoning and coding tasks simultaneously. Detailed statistics are provided in Table 14 and the qualitative analysis is provided in Figures 14, 15 and 16.

**Entropy Analysis** For the entropy analysis, we compute the average entropy over the whole test set for different datasets. In Figure 10, it is clearly visible that for ZENO(NB) scheduler, the average entropy over decoding steps is less and much more stable than the vanilla scheduler. For reasoning tasks, the vanilla scheduler shows a tendency of increasing entropy midway in the generation stage, but for ZENO the entropy over the steps decreases, which points toward better stability, supporting our claim further.

**Visualization for generation window position vs step** In this section, we analyze the generated tokens across steps within the generation canvas. In Figure 11, the top row represents the Vanilla sampler generation for LLaDA. It shows that the vanilla sampler makes an 'X' shape. This is because the vanilla sampler generates from any position in the canvas. Which is the prime contributor to brittle reasoning. On the other hand, ZENO's directional decoding controls its degree of generative freedom, which results in proper output generation. For Dream Instruct, we also observe the same trend 12 but less prominent. Conclusively, in both cases ZENO shows a more stable generation and token reveal pattern, which directly contributes to better generation. **Answer**

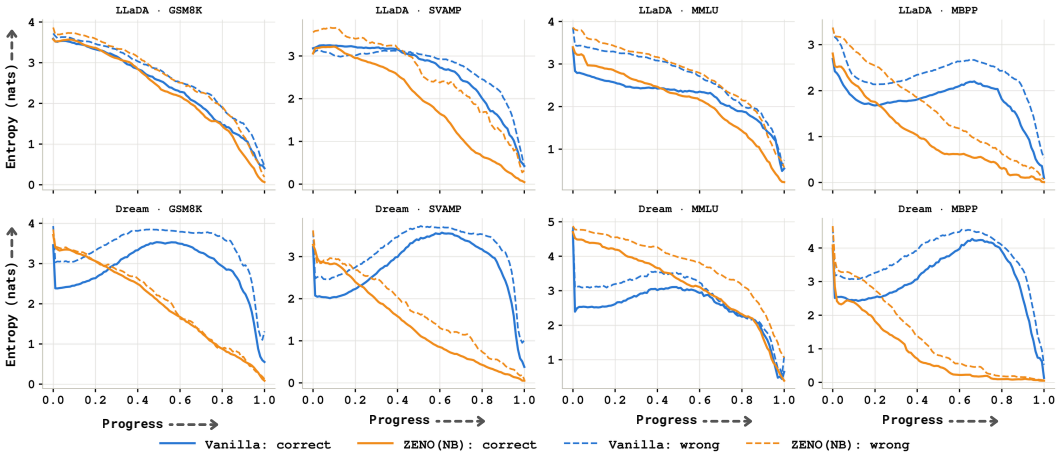



Figure 10: Mean predictive entropy of the still-masked positions vs progress, split by final correctness (solid = correct, dashed = wrong; blue = Vanilla, orange = ZENO(NB))

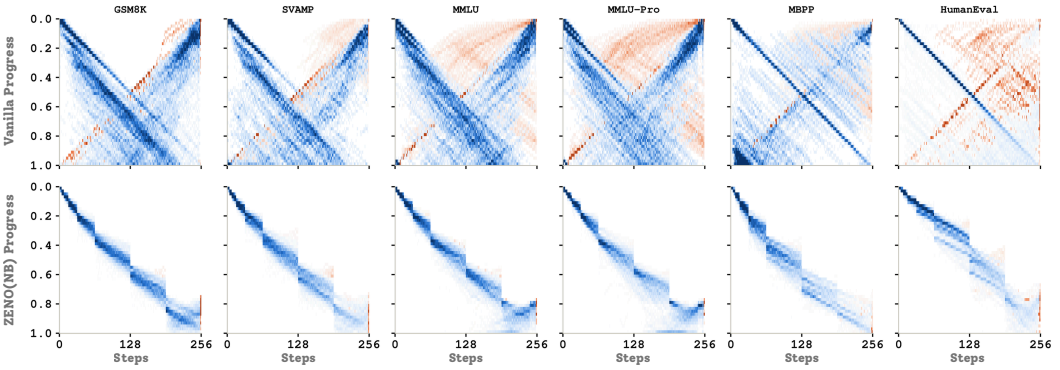



Figure 11: Reveal density for every dataset — LLaDA-8B-Instruct. The y-axis denotes the denoising progression and the x-axis represents the number of denoising steps. Blue indicates content tokens revealed, and orange indicates End-of-Sentence (EoS) or padding revealed.

**Revealed at fraction Reasoning steps:** In this section, we analyze the same output traces to check the answer generation dynamics and validate our claim that ZENO generates proper reasoning and then answers. For this we took the four datasets where we can directly parse the answer. For coding, there is no notion of getting the final answer directly from the generation traces. This analysis reveals that ZENO, across the models and datasets used, generates the reasoning tokens and then generates the answer. For both the LLaDA and Dream model families, ZENO persists in this behavior, and the median is heavily biased towards the right, meaning the answer is generated around the end in most cases. For Vanilla on LLaDA, the answer is generated across the reasoning steps. On Dream, however, Vanilla also generates the answer near the end on several datasets.

**Manual Verification For Low Scoring Baselines:** While analyzing the results, two of the baselines for instruct models were a bit suspicious: the Block diffusion and the Sched baseline. They scored very low for couple of datasets. To ensure the correctness of evaluation, we manually verify the output trajectories and observe that for Block diffusion for zero-shot Instruct model evaluation, it was ending the generation prematurely in some intermediate step and for Sched, it was generating the full trajectory but with the lack of few-shot examples, it was generating completely wrong answers in most of the cases. Apart from this, we manually inspect a subset of output trajectories for all experiments to ensure proper generation and validate the experimental setup. In Table 15 we provide the complete count of correct, incorrect, and answer not generated instances. From the table it is evident that without prior/zero-shot prompting setup ( Figure 9 )for Instruction-tuned models, both Block and Sched samplers fail to follow the simple base instruction

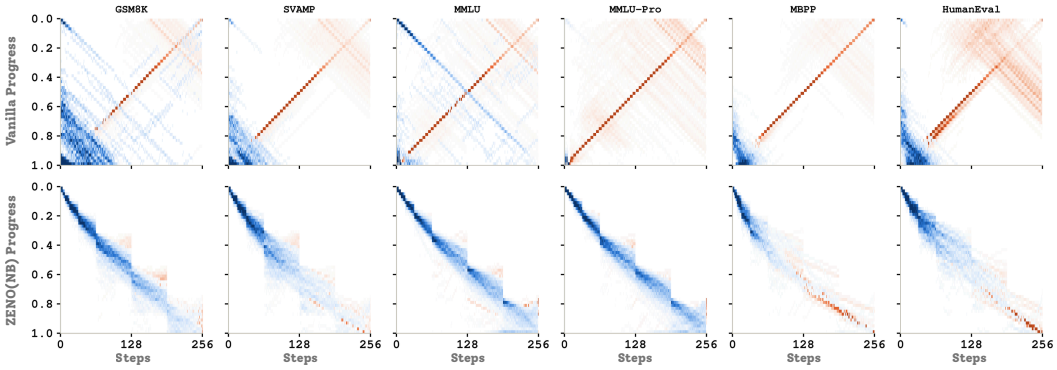



Figure 12: Reveal density for every dataset — Dream-7B-Instruct. The y-axis denotes the denoising progression and the x-axis represents the number of denoising steps. Blue indicates content tokens revealed, and orange indicates End-of-Sentence (EoS) or padding revealed.

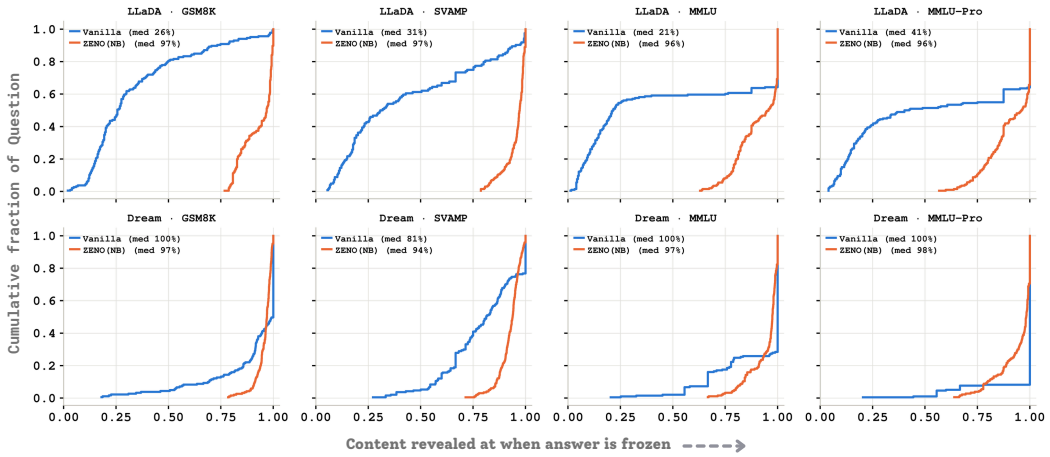



Figure 13: The fraction of content already written when the answer span is frozen, per question. A curve far to the right means the answer comes after the reasoning.

and fail to generate the final answer in many cases. But on the other hand, for the base models, with few-shot instruction tuned prompts, for both the sampler, the number of no-answers generated significantly reduced. Signaling to the fact that, without few-shot prompts or a predefined template, these methods fail. Whereas ZENO outperforms all the samplers in the zero-shot setup itself without explicit prompting.

**On the Significance of ZENO Class of Samplers:** As a concrete verifier of the gains obtained by ZENO, we apply an exact McNemar test on paired per-question outcomes alongside a paired bootstrap with 10,000 resamples on our best variants. Averaged across the LLaDA and Dream instruct models, ZENO(NB) and ZENO(NB)+C outperform the Vanilla baseline by $+12.5$ [11.1, 14.0] and $+11.4$ [9.9, 12.8] points respectively, with both improvements for ZENO(NB) and ZENO(NB)+C highly significant ($p < 10^{-4}$).

## J Hyperparameter Table

Table 16 contains the full list of hyperparameters used throughout all our experiments.

## K Nomenclature

List of all the notations used throughout the paper is provided in Table 17.

| Regime | Model | Dataset | Block | | | Sched | | |
|---|---|---|---|---|---|---|---|---|
| | | | Correct✓ | Wrong× | No ans. | Correct✓ | Wrong× | No ans. |
| **Instruct** | **LLaDA** | GSM8K | 110 | 50 | 240 | 229 | 152 | 19 |
| | | SVAMP | 172 | 27 | 101 | 130 | 48 | 122 |
| | | MMLU | 309 | 257 | 34 | 170 | 159 | 271 |
| | | MMLU-Pro | 117 | 177 | 126 | 105 | 192 | 123 |
| | | MBPP | 114 | 285 | 1 | 111 | 289 | 0 |
| | | HumanEval | 29 | 135 | 0 | 42 | 122 | 0 |
| | **Dream** | GSM8K | 54 | 90 | 256 | 161 | 198 | 41 |
| | | SVAMP | 115 | 15 | 170 | 224 | 55 | 21 |
| | | MMLU | 249 | 135 | 216 | 95 | 88 | 417 |
| | | MMLU-Pro | 89 | 107 | 224 | 24 | 36 | 360 |
| | | MBPP | 114 | 226 | 60 | 160 | 240 | 0 |
| | | HumanEval | 33 | 130 | 1 | 44 | 119 | 1 |
| **Base** | **LLaDA** | GSM8K | 244 | 151 | 5 | 304 | 94 | 2 |
| | | SVAMP | 244 | 54 | 2 | 244 | 55 | 1 |
| | | MMLU | 263 | 195 | 142 | 331 | 257 | 12 |
| | | MMLU-Pro | 123 | 155 | 142 | 166 | 232 | 22 |
| | | MBPP | 119 | 281 | 0 | 165 | 235 | 0 |
| | | HumanEval | 43 | 121 | 0 | 51 | 113 | 0 |
| | **Dream** | GSM8K | 272 | 123 | 5 | 296 | 104 | 0 |
| | | SVAMP | 225 | 75 | 0 | 236 | 64 | 0 |
| | | MMLU | 377 | 197 | 26 | 387 | 196 | 17 |
| | | MMLU-Pro | 166 | 232 | 22 | 185 | 222 | 13 |
| | | MBPP | 215 | 185 | 0 | 226 | 174 | 0 |
| | | HumanEval | 92 | 72 | 0 | 86 | 78 | 0 |

Table 15: Per-question answer breakdown for **Block** and **Sched** on Instruct and Base models, at temperature 0 (greedy) seed 42 (Base is few-shot prompted, Appendix D). **Correct✓** = graded correct; **Wrong×** = an answer was extracted but graded wrong; **No ans.** = no final answer could be extracted from the generation.

**Qualitative Analysis: LLaDA-8B-Instruct (GSM8K)**

**Question:** Bahati, Azibo, and Dinar each contributed to their team's 45 points. Bahati scored the most points and it was 20 more than Azibo scored and 10 more points than Dinar scored. How many points did Azibo score?

**× Vanilla**

To determine how many points Azibo scored, we can set up a system of equations based on the information given. Let's denote the points scored by Azibo as $A$.
According to the problem, the points scored by Bahati is $B = A + 20$, and the points scored by Dinar is $D = B - 10 = (A + 20) - 10 = A - 30$.
The total points scored by the team is 45. Therefore, we can write the equation: $A + B + D = 45$
Substitute the expressions for $B$ and $D$ into the equation:

$$A + (A + 20) + (A - 30) = 45$$

Simplify the equation:

$$A + A + 20 + A - 30 = 45$$

$$3A - 10 = 45$$

$$3A = 45$$

$$A = 15$$

So, Azibo scored $\boxed{15}$ points.

**✓ ZENO (Ours)**

Let's define the points scored by Azibo as $A$. According to the problem, Bahati scored 20 more points than Azibo, so Bahati scored $A + 20$ points. Also, Bahati scored 10 more points than Dinar, so Dinar scored $(A+20)-10 = A+10$ points.
The total points scored by the team is the sum of the points scored by Azibo, Bahati, and Dinar, which is given as 45 points. Therefore, we can write the equation:

$$A + (A + 20) + (A + 10) = 45$$

Simplifying the left side of the equation, we get:

$$3A + 30 = 45$$

To isolate $A$, we subtract 30 from both sides of the equation:
$3A = 15$
Solving for $A$ by dividing both sides by 3:

$$A = 5$$

So, Azibo scored $\boxed{5}$ points.

**Qualitative Analysis: LLaDA-8B-Instruct (MMLU-Pro)**

**Question:** In which of the following representations of numbers by 8-bit words is the addition of the integers 109 and -42 within range?
I. One's complement II. Two's complement III. Sign and magnitude
Options are: (A) I only ; (B) II only ; (C) I and II only ; (D) I, II, and III

**× Vanilla**

To determine which representations allow the addition of the integers 109 and -42 to be within the range of an 8-bit word, we need to consider the properties of each representation.
**I.** One's complement: In one's complement, negative numbers are represented by inverting the bits of the absolute value and adding 1. The sum of 109 and -42 is 67, which is within the range of an 8-bit word.

**II.** Two's complement: In two's complement, negative numbers are represented by inverting the bits of the absolute value and adding 1. The sum of 109 and -42 is 67, which is within the range of an 8-bit word.

**III.** Sign and magnitude: In sign and magnitude, negative numbers are represented by negative values. The sum of 109 and -42 is 67, which is within the range of an 8-bit word.
Therefore, both representations (I and II) allow the addition of the integers 109 and -42 to be within the range of an 8-bit word.

Correct Option is: $\boxed{C}$

**✓ ZENO (Ours)**

To determine which representations of numbers by 8-bit words allow the addition of the integers 109 and -42 to be within range, we need to consider the range of an 8-bit integer, which is from -128 to 127.
I. One's complement: In one's complement representation, the sum of 109 and -42 is 67, which is within the range of an 8-bit integer.

II. Two's complement: In two's complement representation, the sum of 109 and -42 is 67, which is within the range of an 8-bit integer.

III. Sign and magnitude: In sign and magnitude representation, the sum of 109 and -42 is 67, which is within the range of an 8-bit integer.

Therefore, all three representations (one's complement, two's complement, and sign and magnitude) allow the addition of the integers 109 and -42 to be within range.

Correct Option is: $\boxed{D}$

Figure 14: **Two cases where ZENO recovers an answer Vanilla loses. Vanilla** output is shown on the left, ZENO on the right. On the GSM8K problem, Vanilla mis-signs the relation between Bahati's and Dinar's scores and answers 15; ZENO sets the relations up correctly and answers 5. On the MMLU-Pro question about 8-bit integer representations, Vanilla evaluates all three representations and finds each one in range, then names only I and II in its closing sentence and answers (C). ZENO carries all three through to the conclusion and answers (D). Table 14 reports how often this happens.

**Qualitative Analysis: Dream-7B-Instruct (SVAMP)**

**Question:** There are 2 baskets of peaches. Each basket has 4 red peaches and some more green peaches. If there are a total of 12 peaches in all baskets How many green peaches are in each basket?

**× Vanilla**

Let's denote the number of green peaches in each basket as $G$.
Each basket has 4 red peaches, so both baskets have 4 + 4 = 8 red peaches.
Since the total number of peaches in both baskets is 12, we can write the equation:

$$8 + G = 12$$

Now, we solve for $G$:

$$G = 12 - 8$$
$$G = 4$$

So, each basket has $\boxed{4}$ green peaches.

**✓ ZENO (Ours)**

Let's denote the number of green peaches in each basket as $g$.
Since each basket has 4 red peaches, the total number of red peaches in both baskets is

$$4 + 4 = 8$$

.
The total number of peaches in both baskets is given as 12.
Therefore, the number of green peaches in both baskets is $12 - 8 = 4$.
Since there are two baskets, the number of green peaches in each basket is $\frac{4}{2} = 2$.
So, there are $\boxed{2}$ green peaches in each basket.

**Qualitative Dream-7B-Instruct (MMLU-Pro)**

**Question:** Mars has an atmospheric pressure of 600 pascals. If the acceleration due to gravity on Mars is (roughly) 4 m/s2 and the surface area of Mars is about 145000000 square km what is the approximate mass of the martian atmosphere?
Options are: (A) 2.2x1011 kg ; (B) 2.2x1014kg ; (C) 2.2x1020kg ; (D) 2.2x1016 kg

**× Vanilla**

145000000 square km = 145000000 x 10^6 square meters = 145 x 10^12 square meters
Now we can calculate the force: Force = Pressure x Area

Force = 600 Pa x 145 x 10^12 m^2 = 8.7 x 10^14 N
Finally, we can find the mass:
Mass = Force / Gravity

Mass = 8.7 x 10^14 N / 4 m/s^2

Mass ≈ 2.2 x 10^14 kg

Therefore, the mass of the martian atmosphere is approximately 2.2 x 10^14 kg.
Correct Option is: $\boxed{B}$

**✓ ZENO (Ours)**

To find the mass of the martian atmosphere, we can use the formula for pressure, which is:
Pressure (P) = Force (F) / Area (A)
The force (F) is equal to the mass (m) times the acceleration due to gravity (g), so we can rewrite the formula as:
P = m * g / A
Rearranging the formula to solve for mass (m), we get:
m = P * A / g
Now we can plug in the given values:
P = 600 pascals A = 145000000 square km = 145000000 * 10^6 square meters g = 4 m/s^2
m = 600 * (145000000 * 10^6) / 4 m = 600 * 145000000 * 10^6 / 4 m = 2.2 * 10^16 kg
So the approximate mass of the martian atmosphere is 2.2x1016 kg.
$\boxed{D}$

Figure 15: **Two further cases where ZENO recovers an answer Vanilla loses. Vanilla** output is shown on the left, ZENO on the right. On the SVAMP problem, Vanilla computes the 4 green peaches across both baskets but reports that figure as the per-basket count; ZENO divides by the number of baskets and answers 2. On the MMLU-Pro problem, Vanilla converts square kilometres to square metres correctly, then drops two orders of magnitude when multiplying pressure by area, writing $8.7 \times 10^{14}$ N for $8.7 \times 10^{16}$ N. It answers (B); ZENO keeps the magnitude through both steps and answers (D).

**Qualitative Analysis: Failure case SVAMP LLaDA-8B-Instruct**

**Question:** For the walls of the house he would use 11 nails in all to secure large planks of wood. If each plank needs 3 pieces of nails to be secured and an additional 8 nails were used. How many planks does John need for the house wall?

**✓ Vanilla**

To determine how many planks John needs for the house wall, we can follow these steps:
1. **Identify the total number of nails used:** John used 11 nails in total.
2. **Determine the number of nails used for planks:** We know that an additional 8 nails were not used for the planks. Therefore, the number of nails used for the planks is:

$$\text{Nails used for planks} = \text{Total nails} - \text{Additional nails}$$
$$= 11 - 8 = 3$$

3. **Determine the number of nails needed per plank:** Each plank requires 3 pieces of nails to be secured.
4. **Calculate the number of planks:** To find the number of planks, we divide the number of nails used for planks by the number of nails needed per plank:

$$\text{Number of planks} = \frac{\text{Nails used for planks}}{\text{Nails per plank}} = \frac{3}{3} = 1$$

Therefore, John needs $\boxed{1}$ plank for the house walls.

**× ZENO (Ours)**

To determine how many planks John needs for the house wall, we can follow these steps:

1. Identify the total number of nails used. 2. Identify the number of nails used per plank. 3. Calculate the number of planks by dividing the total number of nails by the number of nails used per plank.

Step 1: Identify the total number of nails used. The total number of nails used is 11.
Step 2: Identify the number of nails used per plank. Each plank requires 3 pieces of nails.
Step 3: Calculate the number of planks by dividing the total number of nails by the number of nails used per plank.

$$\text{Number of planks} = \frac{\text{Total number of nails}}{\text{Number of nails per plank}}$$
$$= \frac{11}{3} = 3.7$$

Therefore, John needs $\boxed{3}$ planks for the house wall.

**Qualitative Analysis: Failure case GSM8K Dream-7B-Instruct**

**Question:** Martha is planning her Christmas party. She invited 2 families with 6 people and 3 families with 4 people. 8 people couldn't come due to illness, and 1/4 that number had previous commitments. How many people show up for Martha's party?

**✓ Vanilla**

Martha invited 2 families with 6 people and 3 families with 4 people, for a total of 2*6 + 3*4 = 12 + 12 = 24 people. 8 people couldn't come due to illness, and 1/4 that number had previous commitments, so 8 + 2 = 10 people couldn't come. Therefore, 24 - 10 = $\boxed{14}$.

**× ZENO (Ours)**

Martha invited 2 families with 6 people, so that's 2 * 6 = 12 people. She also invited 3 families with 4 people, so that's 3 * 4 = 12 people. In total, Martha invited 12 + 12 = 24 people. 8 people couldn't come due to illness, so that's 24 - 8 = 16 people. 1/4 of that number had previous commitments, so that's 8 people. Therefore, 16 - 8 = 8 people show up for Martha's party. $\boxed{8}$

Figure 16: **Two cases in the opposite direction, where Vanilla is correct and ZENO is not. Vanilla** output is shown on the left, ZENO on the right. On the SVAMP problem (LLaDA), Vanilla removes the 8 additional nails before dividing ($11 - 8 = 3$ nails, $3 \div 3 = 1$ plank); ZENO divides the full nail count by the nails per plank and answers 3. On the GSM8K problem (Dream), Vanilla applies the one-quarter to the 8 absentees, giving 2 more, and answers $24 - 10 = 14$. ZENO applies it to the 16 remaining after the illness instead, reports that quarter as 8, and answers $16 - 8 = 8$.

| Parameters Type | Sub-Category | Parameter | Value |
|---|---|---|---|
| Decoding | Budget | max_new_tokens ($L$) | 256 |
| | | steps ($S$) | 256 |
| | | gen_len_grid (ablation) | {256, 512} |
| | | steps_grid (ablation) | {256, 512} |
| | Sampling | temperature ($\tau_{\text{gen}}$) | 0 (greedy) |
| | | temperature (seed study) | 0.7 |
| | | guidance scale ($s_{\text{cfg}}$) | 0.0 |
| ZENO | Window schedule | initial width ($w_0$)–ZENO (N) | 1 |
| | | initial width ($w_0$) | 4 |
| | | growth base ($\gamma$) | 2 |
| | | skip factor ($\kappa$) | 0 |
| | | switch width ($W^\star$) | 128 |
| | | post-switch block length ($\beta$) | 64 |
| | Reveal rule | confidence threshold ($\tau$) | 0.80 |
| | | per-step cap ($M$) | 64 |
| Block | Baseline (Block) | block length ($B_{\text{blk}}$) | 64 |
| | | max block-fill length | 256 |
| | | Number of blocks | 4 |
| Execution | Runtime | dtype | bfloat16 |
| | | batch size (accuracy tables) | 1 |
| | | batch size (timing tables) | 1 |
| | | GPU memory fraction | 0.90 |
| | Reproducibility | seed (greedy sweeps) | 42 |
| | | seed grid (seed study) | 42, 123, 512, 1024, 4096 |
| | Compute | GPU | NVIDIA A100-80GB |
| | | GPU count | 8× (1 node) |
| Random Walk | Simulation (App. A) | dimensions ($\rho$) | $\{1, \ldots, 5\}$ |
| | | bias levels ($v$) | $\{0.5, 0.6, 0.7, 1.0\}$ |
| | | walks per cell | 20,000 |
| | | step budget ($S_{\text{rw}}$) | 512 |
| | | hit radius ($R$) | 20 |
| | | target distance ($D$) | 50 |

Table 16: Table containing all the hyperparameter configurations used in the paper.

| Notation | Description |
|---|---|
| $T$ | Total sequence length: prompt length plus generation length. |
| $P_r$ | Prompt length – number of conditioning tokens. |
| $L$ | Generation length – size of the masked generation region. |
| $x^{(t)}$ | Canvas – the partially masked sequence at step $t$. |
| $\mathcal{G}$ | Generation region – the $L$ positions to be filled. |
| $\mathcal{M}^{(t)}$ | Mask set – positions still masked at step $t$. |
| $B_{\text{blk}}$ | Block length of the Block baseline; Vanilla uses one block of length $L$. |
| $\mathcal{V}$ | Vocabulary – the token vocabulary of the model. |
| $f_\theta$ | The frozen diffusion language model. |
| $\Delta_m$ | Phase increment – positions newly exposed by phase $m$. |
| $\gamma$ | Growth base – multiplicative window growth per phase. |
| $w_0$ | Initial width – size of the first window. |
| $w_m$ | Window width – zone size in phase $m$. |
| $\mathcal{Z}_m$ | Zone – generation positions open in phase $m$. |
| $\tilde{c}_i^{(t)}$ | Masked confidence – $c_i^{(t)}$ inside the zone, $-\infty$ outside. |
| $S$ | Step budget – nominal number of denoising steps. |
| $S'$ | Phase budget – steps allotted per window, $\lceil S/K \rceil$. |
| $K$ | Phase count – number of windows, $\lvert\mathcal{W}^{\text{N}}\rvert$ (derived). |
| $\kappa$ | Skip factor – number of intermediate powers of $\gamma$ skipped. |
| $\beta$ | Post-switch block length – window increment after the switch width. |
| $\mathcal{E}^{(t)}$ | Eligible set – positions the sampler may commit at step $t$. |
| $c_i$ | Confidence – the model's top-1 probability at position $i$. |
| $M$ | Per-step cap – maximum positions committed in one pass. |
| $R_t$ | Reveal set – positions committed at step $t$. |
| $\tau_{\text{gen}}$ | Temperature – Gumbel noise scale; 0 is greedy decoding. |
| $s_{\text{cfg}}$ | Guidance scale – classifier-free guidance strength. |
| $\ell_{\text{cond}}^{(t)}$ | Conditioned logits – forward pass with the prompt/context visible. |
| $\ell_{\text{uncond}}^{(t)}$ | Unconditioned logits – forward pass with the context re-masked. |
| $g_{i,v}$ | Gumbel noise – i.i.d. noise used for Gumbel-max sampling. |
| $\ell_i^{(t)}$ | Logits – the model's output at position $i$. |
| $\hat{x}_i^{(t)}$ | Candidate – the token drawn at position $i$ before committing. |
| $k_t$ | Transfer count – positions to commit at step $t$ under top-$k$ reveal. |
| $\mathcal{W}^{\text{N}}$ | Geometric schedule – pure geometric window growth. |
| $\mathcal{W}^{\text{NB}}$ | Hybrid schedule – geometric growth up to $W^\star$. |
| $\tau$ | Threshold – confidence required to commit a token (all reported variants). |
| $W^\star$ | Switch width – window size past which growth becomes block-wise. |
| $\rho$ | Dimension of the random walk (number of axes). |
| $A$ | Starting point of the walker, fixed at $\mathbf{0}$. |
| $B$ | Target point the walker attempts to reach. |
| $D$ | Distance between $A$ and $B$. |
| $a_t$ | Axis selected at step $t$, drawn uniformly from $\{0, \ldots, \rho - 1\}$. |
| $v$ | Bias probability. |
| $S_{\text{rw}}$ | Step budget of the random walk (distinct from $S$). |
| $T_{\max}$ | Time horizon over which a hit is checked (distinct from $T$). |
| $R$ | Hit radius – a walker counts as a hit if it ever comes within $R$ of $B$. |

Table 17: Collection of symbols used in the whole paper.