BlockBatch: Multi-Scale Consensus Decoding for Efficient Diffusion Language Model Inference

Anonymous ACL submission

Abstract

Diffusion language models (dLLMs) generate text by iteratively denoising multiple token positions in parallel, offering an attractive alternative to strictly autoregressive decoding. In practice, however, block-wise dLLM inference exposes a difficult granularity trade-off: small blocks preserve local conditioning but require many denoising steps, whereas large blocks expose more parallelism but can make premature commitments and accumulate cache error. Existing acceleration methods typically choose a single block size per request, leaving the complementarity among block sizes unused. We show that block size itself is a useful branching dimension. Different block sizes induce related but non-identical KV-cache trajectories: branches often share an initial prefix, bifurcate at semantically decisive positions, and later agree on syntactically lightweight tokens. Motivated by this structure, we propose BlockBatch, a training-free online inference framework that executes multiple block-size branches for the same request inside a batched forward pass. BlockBatch coordinates these branches through confidence-gated token merging, leader-based synchronization, and periodic full-sequence refreshes that re-anchor local block updates to a globally consistent KV state. Across 3 representative dLLMs and 4 datasets, BlockBatch reduces denoising NFEs by 26.6% on average and achieves a 1.33× average end-to-end speedup over Fast-dLLM while preserving accuracy. These results identify block-size diversity as a practical and previously underexplored axis for branch-parallel dLLM inference.

1 Introduction

Diffusion language models (dLLMs) (Gong et al., 2025) replace left-to-right factorization with iterative masked denoising (Campbell et al., 2022, 2024; Chen et al., 2023; Vignac et al., 2023): at each generation round, the model predicts distributions over many token positions and commits a subset of tokens in parallel. Recent masked dLLMs (Ye et al., 2025a; Nie et al., 2025b) have substantially narrowed the gap to autoregressive language models and, in several regimes, exhibit competitive scaling, instruction-following, and flexible generation behavior (Sahoo et al., 2024c; Nie et al., 2025b; Ye et al., 2025b; Prabhudesai et al., 2025). This parallelism is attractive for inference, but it is not free. Committing too few tokens preserves local conditioning but requires many denoising rounds; committing too many tokens increases apparent parallelism but can make premature decisions and degrade cache consistency (Ma et al., 2025). Efficient dLLM inference hence depends on finding the right granularity at which to trade off parallel token updates, confidence, and state reuse. A central mechanism for making this trade-off practical is block-wise inference with approximate KV reuse (Jiang et al., 2026). Instead of recomputing the full sequence at every denoising step, block-wise methods update a local decoding window while reusing, delaying, or selectively refreshing cached KV states outside the active region (Wu et al., 2026; Liu et al., 2025; Jiang et al., 2026). The block size determines the granularity

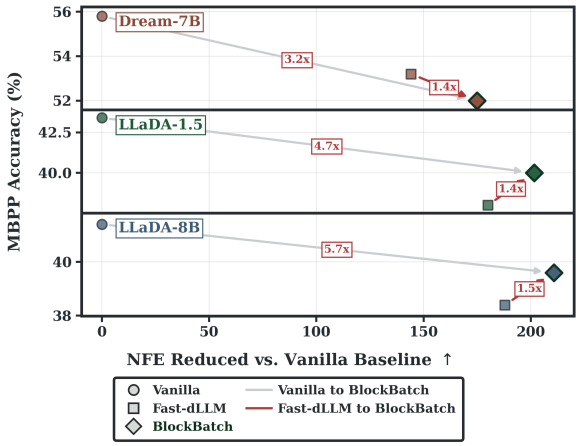


Figure 1: **BlockBatch speedup against Fast-dLLM and the vanilla baseline on MBPP.**

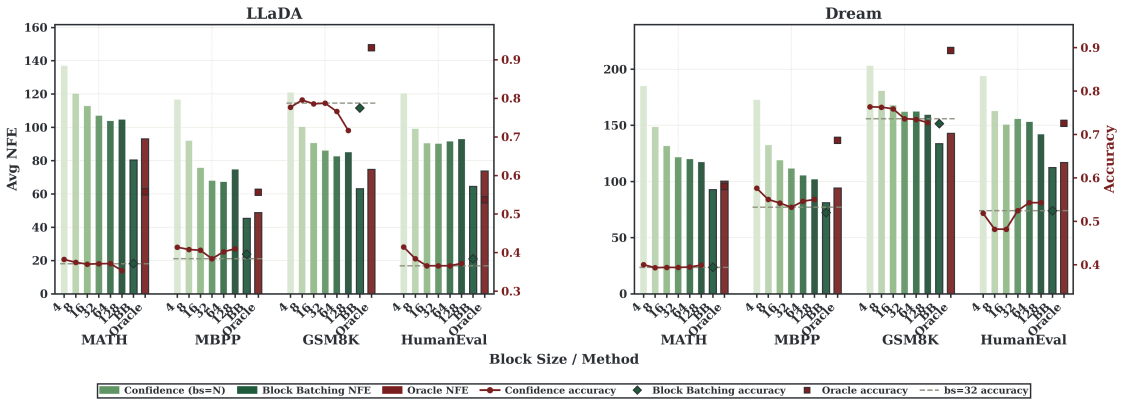


Figure 2: **Per-sample block-size oracle versus fixed block-size decoding.** Bars report average NFE and curves/markers report accuracy for fixed confidence-decoding block sizes, block batching, and an oracle that selects the best block size per prompt. The oracle improves the accuracy–NFE tradeoff across models and tasks, motivating block size as a per-sample branching dimension rather than a fixed hyperparameter.

of this process. Small blocks are conservative: they more closely approximate sequential conditioning but require more denoising function evaluations (NFEs) (Cheng et al., 2025). Large blocks expose more parallel work per model invocation, but they weaken intra-block conditioning, accumulate larger cache discrepancies, and may require downstream correction (Nie et al., 2025a; Zhang et al., 2025). Existing systems therefore typically select a single block size for a request, either as a fixed hyperparameter or through a local adaptation rule. This single-trajectory view leaves a natural opportunity unused: different block sizes may be useful for different parts of the same generation.

Our starting point is that block size is not merely an inference hyperparameter, but a source of complementary decoding trajectories. In our experiments shown in Fig. 2, a per-sample oracle that chooses among block sizes achieves a better accuracy–NFE tradeoff than any single fixed block-size configuration. This indicates that there is no globally optimal block size across prompts, tasks, or models. The observation suggests a different form of branch-parallel decoding. Prior acceleration methods for autoregressive language models often branch along the token-candidate axis, producing speculative or tree-structured continuations that are later verified, accepted, or discarded (Leviathan et al., 2023; Cai et al., 2024a). In contrast, we branch along the block-size axis. All branches share the same prompt and initial KV cache, but they advance with different denoising granularities and therefore trace different trajectories through the space of partially decoded sequences and KV-cache states.

These block-size branches are neither independent generations nor redundant copies. At the token level, they often agree on an initial prefix, bifurcate at a small number of semantically decisive positions, and later re-agree at positions with low local ambiguity. Our token-category analysis shows that such later-stage agreement is concentrated on syntactically lightweight tokens (see Appendix B.1). In contrast, bifurcation positions tend to carry the semantic content that determines the final answer. At the KV-cache level, block denoising acts as a local update: it modifies only the active region while relying on cached context elsewhere, where a full-sequence refresh re-anchors the branch to a globally consistent KV state. Together, these observations suggest that useful multi-branch dLLM inference should preserve branch diversity around bifurcation points, exchange information only when branches are compatible, and periodically repair stale cache states.

Motivated by the above findings, we propose BlockBatch, a training-free online inference framework for efficient dLLM decoding. For each request, BlockBatch instantiates multiple branches with different block sizes and executes their active windows inside a single batched model invocation. The branches are coordinated by three important operations. First, a confidence-gated merge transfers token proposals only between compatible branches and only when the destination branch assigns sufficient probability to the proposed token. Second, a leader-based synchronization rule prevents lagging branches from wasting computation by copying the sequence state and KV cache of a sufficiently advanced branch. Third, periodic full-sequence refreshes recompute KV states for active branches, correcting accumulated drift from

local block updates. Large-block branches provide aggressive progress, small-block branches provide conservative alternatives, and merge/sync/full-sequence refresh operations prevent the search from degenerating into unrelated or stale trajectories. In summary, our contributions are as follows:

- **Block-size diversity as branch parallelism.** We identify block size as an underexplored axis for multi-branch dLLM inference and show that different block sizes provide complementary accuracy–efficiency tradeoffs across samples.
- **Token- and KV-level characterization.** We analyze how block-size branches relate to one another, revealing token-level bifurcation and later-stage consensus as well as KV-cache dynamics in which local block updates are periodically corrected by full-sequence refreshes.
- **A training-free inference framework.** We introduce BlockBatch, which executes multiple block-size branches in parallel and coordinates them through confidence-gated merging, leader-based synchronization, and full-sequence KV refresh.
- **Empirical validation.** Experiments on three dLLMs (LLaDA-1.5-8B, LLaDA-Instruct-8B, Dream-7B) across four benchmarks (GSM8K, MATH, HumanEval, MBPP) show that BlockBatch reduces denoising steps by 26.6% and achieves a 1.33× end-to-end speedup on average over Fast-dLLM while preserving accuracy.

2 Preliminaries

2.1 Diffusion LLMs as a Discrete Markov Process

Unlike continuous diffusion models that operate on Gaussian noise (Sohl-Dickstein et al., 2015a; Song and Ermon, 2019; Ho et al., 2020a; Song et al., 2021a,b), Masked Diffusion Language Models (MDLMs) operate in a discrete, categorical state space (Hoogeboom et al., 2021a; Austin et al., 2021a; Sahoo et al., 2024a; Nie et al., 2025b; Sohl-Dickstein et al., 2015b; Hoogeboom et al., 2021b). Let $\mathcal{V}$ denote the vocabulary and let [M] denote the special [MASK] token. The extended vocabulary is $\tilde{\mathcal{V}} = \mathcal{V} \cup \{\texttt{[M]}\}$. For a sequence of length L, the state at generative step t is represented as $\mathbf{X}_t \in \tilde{\mathcal{V}}^L$.

The generative reverse process transitions from a fully masked state $\mathbf{X}_T = [\texttt{[M]}, \ldots, \texttt{[M]}]$ to a decoded state $\mathbf{X}_0 \in \mathcal{V}^L$ (Sahoo et al., 2024b; Ho et al., 2020b). At each step t, the network observes $\mathbf{X}_t$ and outputs a categorical distribution $p_\theta(x^{(i)} = v \mid \mathbf{X}_t)$ for each position i. Standard diffusion LLMs decode this distribution using a confidence-based mechanism. The top-1 confidence score is $c_{t,i} = \max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{X}_t)$. Given a confidence threshold τ_{conf} and the most confident masked index i^*, the deterministic transition for position i is

$$x_{t-1,i} = \begin{cases} \arg\max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{X}_t), & \text{if } x_{t,i} = \texttt{[M]} \text{ and } (c_{t,i} \geq \tau_{\text{conf}} \vee i = i^*), \\ x_{t,i}, & \text{otherwise.} \end{cases} \tag{1}$$

We can formalize the overall transition probability as $P(\mathbf{X}_{t-1} \mid \mathbf{X}_t)$. The sequence therefore forms a discrete-time Markov chain:

$$P(\mathbf{X}_{t-1} \mid \mathbf{X}_t, \mathbf{X}_{t+1}, \ldots, \mathbf{X}_T) = P(\mathbf{X}_{t-1} \mid \mathbf{X}_t). \tag{2}$$

2.2 KV Cache in Diffusion LLMs

For batched inputs, prefix caching provides the first source of reuse by sharing the KV states of identical prompt prefixes across requests. This reuse pattern is standard in autoregressive Transformer serving, where KV caching avoids recomputing previous key–value activations during decoding (Vaswani et al., 2017; Shazeer, 2019; Dao et al., 2022; Kwon et al., 2023). Modern LLM serving systems further improve prefix reuse and cache sharing through paged cache allocation, radix-tree or prefix-tree cache layouts, and shared-prefix attention kernels.

Under the semi-autoregressive decoding paradigm, Fast-dLLM (Wu et al., 2025b) introduces block-wise approximate KV caching: previously decoded blocks are reused, while the current block is recomputed during denoising. Several follow-up methods refine which cache entries should be recomputed and which should be retained across steps. dLLM-Cache (Liu et al., 2025) observes that dLLM inference contains a mostly static prompt and a partially dynamic response, and applies long-interval prompt caching together with feature-similarity-guided response updates. dKV-Cache exploits token-dependent representation dynamics through delayed KV reuse, storing decoded-token KV states one step later and providing decode and greedy variants that trade quality for speed (Ma et al., 2025). d^2Cache selects masked tokens using confidence and a

certainty prior, then selects prompt and decoded tokens using attention-aware importance before updating only the selected KV states (Jiang et al., 2026). Related dLLM acceleration methods also exploit guided unmasking, local determinism, block-diffusion schedules, or hierarchical caching to reduce denoising cost (Hu et al., 2025; Kong et al., 2025a; Fu et al., 2025; Wu et al., 2025a).

More broadly, these dLLM cache designs are connected to a larger body of KV-cache management work in autoregressive LLM inference. Prior systems retain important tokens, evict or compress stale cache entries, quantize KV states, or fetch only query-relevant cache pages (Zhang et al., 2023; Liu et al., 2023; Xiao et al., 2024; Li et al., 2024; Cai et al., 2024b; Tang et al., 2024; Liu et al., 2024; Lee et al., 2024). These methods can be viewed through a common systems lens: selecting which KV states to reuse and which to refresh, then balancing stale and refreshed cache states to trade accuracy against latency.

3 KV-Level and Token-Level Characterizations of dLLM Inference

3.1 Block-Size Diversity as a Branching Axis

Existing dLLM inference systems pick a single block size per request, treating it as a hyperparameter. However, this design choice is suboptimal, as the accuracy-maximizing block size varies substantially across prompts. To quantify this, we evaluate a *per-sample oracle* that selects, for each prompt, the block size in $\{4, 8, 16, 32, 64, 128\}$ that yields the highest accuracy under confidence-based decoding in Fig. 2. The oracle is not realizable at inference time, but it upper-bounds any sample-wise block-size selection policy built on top of confidence decoding.

Fig. 2 reports NFE and accuracy on MATH (Hendrycks et al., 2021a) and MBPP (Austin et al., 2021b) for LLaDA-Instruct-8B (Nie et al., 2025b) and Dream-Base-7B (Ye et al., 2025a). The oracle outperforms every fixed block size on both models and tasks while using fewer NFEs than the smaller block-size configurations, confirming that block-size diversity carries genuine per-sample information. These observations motivate treating **block size** as a **branching dimension** rather than a hyperparameter. Notably, BlockBatch achieves even lower NFE while preserving accuracy through a per-step policy of sync and merge, which is a finer granularity version of the per sample policy, demonstrating that branching dimension is still an exploitable dimension.

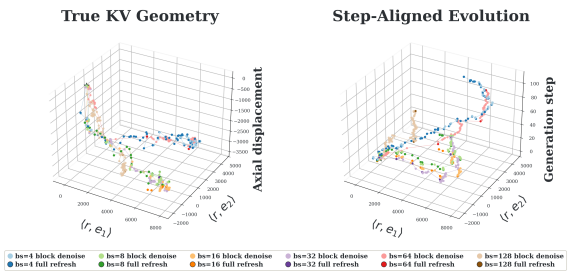


Figure 3: **KV-cache trajectory diagnostic on `HumanEval` sample 3.** Both panels visualize logged KV-cache states in the same tangent coordinate system around the prompt anchor c_0. The horizontal axes are the tangent projections $(\langle r, e_1 \rangle, \langle r, e_2 \rangle)$, where e_1 and e_2 are the leading SVD directions of the centered residuals. **Left: True KV Geometry.** The vertical axis shows axial displacement from the prompt anchor. As generation progresses, different block-size branches bifurcate and spread across the tangent plane, supporting the tangent-space bifurcation behavior described in Proposition 3 / Sec. C.4. **Right: Step-Aligned Evolution.** The vertical axis is replaced by generation step, revealing the temporal structure of the same trajectories. Adjacent block-denoise updates remain locally clustered, whereas large trajectory jumps are introduced by full-sequence refreshes. This pattern indicates that refresh operations, rather than local block denoising alone, are the main source of large KV-cache trajectory displacement, consistent with Proposition 1 / Sec. C.2.

3.2 KV-Level Characterization: KV-Cache Trajectory Diagnostics

While Fig. 2 shows that block size is a useful branching axis, the block-size parameter itself is not the final explanation for the observed differences. Instead, we show that the underlying mechanism through which block size changes NFE and accuracy is the trajectory induced in KV-cache space.

Different block sizes change how many token positions are updated under a shared cached context, how long a branch relies on local cache reuse, and how frequently the branch is re-anchored by a full-sequence refresh. Thus, the effect of block size is expressed through the sequence of KV-cache states visited during inference.

Let $K_t^{(b)}$ denote the flattened KV-cache state of branch b at event t. At the KV level, block-wise dLLM inference alternates between a *block-denoise* update, $K_{t+1}^{(b)} = K_t^{(b)} +$

$\Delta_{\text{blk}}(K_t^{(b)}, x_t^{(b)}, B_t^{(b)})$, and a *full-sequence refresh*, $K_{t+1}^{(b)} = F(x_t^{(b)})$. In the first operation, only the active block is recomputed while the model attends to cached KV states outside the current window. In the second operation, the KV cache is recomputed from the entire current sequence. Appendix C formalizes these operations. In particular, Proposition 1 (Sec. C.2) shows that block-denoise updates should produce smaller local movements in KV-cache vector space than full-sequence refreshes, because the direct update is restricted to the active block.

This distinction is visible in Fig. 3. The projected trajectories show that adjacent block-denoise steps move locally within a compact region of the KV space, whereas full-sequence refreshes produce larger global corrections between clusters. This supports the interpretation that block-size branches are not independent generations but rather different local traversals of a shared KV-cache geometry. Larger blocks tend to make more aggressive local progress under reused cache states, while smaller blocks follow more conservative trajectories with finer-grained conditioning.

The same view also explains why periodic refresh is necessary. A block-denoise step is efficient but may increase this error because only a local window is updated. A full-sequence refresh is more expensive but contracts the error by recomputing the cache from the full token state. Proposition 2 (Sec. C.3) shows that if a refresh is applied every R block-denoise steps, the cache-consistency error remains bounded when $\beta(1+\lambda_b)^R < 1$. This gives a KV-space explanation for the refresh interval used by BlockBatch: full-sequence refreshes are not merely an implementation detail, but rather the mechanism that prevents local block updates from drifting too far from the model-consistent KV manifold.

3.3 Token-Level Characterization: Bifurcation Tokens and Later-Stage Consensus

During block-batched inference, branches initialized with different block sizes begin from the same prompt and prefill cache, so their early decoded tokens often remain identical (though the length of the shared prefill varies across block sizes; see Tab. 5 in Appendix B). Divergence typically occurs at specific *bifurcation tokens*: once two branches commit different tokens at the same position, their subsequent token distributions and KV-cache trajectories begin to separate, since later predictions are conditioned on the current partially decoded sequence and its associated cache state.

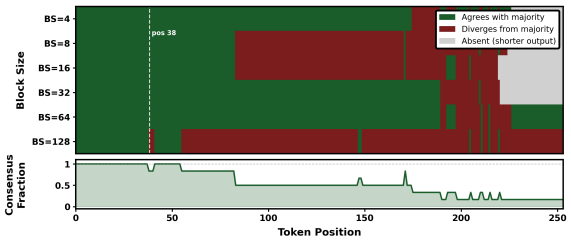


Figure 4: Block-size branch consensus on HumanEval sample 3. Branches share an identical prefix for 37 tokens, after which BS = 128 bifurcates first and smaller blocks split off later. Isolated green columns inside red regions (e.g. pos 43, 150, 170) are later-stage consensus events — previously diverged branches briefly re-agree on a token while their KV trajectories remain distinct, as shown in Fig. 3.

Interestingly, this token-level divergence does not always persist monotonically. Even after an early bifurcation, branches may later produce the same token at the same position. We refer to this phenomenon as *later-stage consensus*. As shown in Fig. 4, later-stage consensus appears around token position 190, highlighted in a deeper green.

However, later-stage consensus should not be interpreted as full trajectory convergence. Agreement on a later token only indicates that different branches selected the same discrete token at that position. Their underlying KV states, token histories, and conditional distributions may still differ substantially. In this sense, it is a token-level agreement event rather than a guarantee of KV-level convergence. This distinction is important: branches can agree on individual high-confidence tokens while still following different inference trajectories in KV-cache vector space.

4 Method

Inspired by the characterizations in Sec. 3, we observe that the main opportunity in block-wise dLLM inference is not to choose a single globally optimal block size, but to exploit the complementary trajectories induced by multiple block sizes during the same decoding process. We therefore propose **BlockBatch** (Fig. 5), a training-free framework that runs multiple block-size branches for a single request in one batched forward pass. Each branch alternates cheap local *block denoising* with periodic *full-sequence refreshes* that re-anchor them to a consistent KV state, while a confidence-gated *merge* and leader-based *sync* operations exchange compatible tokens and prune stale branches

around bifurcation points.

4.1 Exploiting Bifurcation

Block-wise denoising may enter low-progress regions where local updates do not advance enough tokens. Instead of treating multiple branches only as redundant work, we use their diversity as a source of useful proposals. Branches can share tokens and KV states when their decoded prefixes are compatible. The detailed algorithm is shown in Algorithm 2 (Appendix D), where the overview is shown as Fig. 5.

Confidence-Gated Merge The merge operation transfers high-confidence token proposals across compatible branches. A source branch is compatible with a destination branch if all positions decoded in both branches are exactly the same. For a masked position in the destination branch, we inspect tokens proposed by compatible source branches and accept the best candidate only when the destination branch's own probability map assigns that token confidence above a fixed threshold. This gives lagging branches free progress without forcing them to accept tokens inconsistent with their local state.

Leader-Based Sync The sync operation handles larger progress gaps. If the leading branch has decoded more than a threshold number of tokens beyond a lagging branch (this number is controlled by the sync threshold, see Appendix A for ablation Tab. 2), the lagging branch copies the leader's sequence state and KV cache row. Its block window is then realigned to the first remaining masked position. Sync therefore acts as a controlled reset: it preserves the fastest reliable trajectory while preventing slow branches from wasting computation in stale regions of the KV space.

4.2 System-Level Optimization

Fig. 6-(a) shows per-step block denoising latency on an H200 across varying batch size $\times$ block size combinations. Latency stays nearly flat in the memory-bound regime but rises sharply once the total token count exceeds roughly 256, where denoising becomes compute-bound. We exploit this so that multi-branch denoising costs about the same as denoising a single branch.

BlockBatch Denoise Optimization. At each denoise step, only the unmasked positions in each branch's current block produce queries. We pack these queries from all branches into a single variable-length buffer and issue one forward pass, supplying the cumulative offsets that FlashAttention's varlen kernel requires. The same packed tensor flows through every transformer block, so both attention and the FFN operate on this compact buffer end-to-end (Fig. 6-(b)).

5 Experiments

5.1 Setup

We evaluate BlockBatch on LLaDA-1.5-8B, LLaDA-Instruct-8B (Nie et al., 2025b), and Dream-Base-7B (Ye et al., 2025b). We compare against Vanilla diffusion decoding, Fast-dLLM dual cache (Wu et al., 2026), and LocalLeap (Kong et al., 2025b). Unless stated otherwise, all methods generate up to 256 tokens with batch size 1 and confidence threshold 0.9. For Fast-dLLM dual cache, we use block size 32. For LocalLeap, we use the default script settings: anchor threshold 0.9, radius 4, and relaxed threshold 0.75 for LLaDA and 0.8 for Dream. For BlockBatch, we run block sizes $\{4, 8, 16, 32, 64, 128\}$ in one fused generation pass. Each branch owns an independent sequence row and KV-cache row; batching only changes execution shape. We count one batched model forward as one NFE: $\mathrm{NFE}_{\mathrm{total}} = \mathrm{NFE}_{\mathrm{init}} + \mathrm{NFE}_{\mathrm{block}} + \mathrm{NFE}_{\mathrm{refresh}}$. We evaluate on GSM8K (Cobbe et al., 2021a), MATH (Hendrycks et al., 2021a), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021b). Accuracy is computed with the official task-specific evaluation logic; for code tasks, generated code is postprocessed with the model-specific sanitization pipeline before execution-based evaluation. Additional prompt, baseline, and ablation settings are provided in Appendix E.3.

5.2 Main Results

Tab. 1 reports latency, NFE, and accuracy across twelve model–task settings. The main result is that BlockBatch is the most NFE-efficient method in every setting: it achieves the lowest or tied-lowest NFE across all twelve comparisons, with 47.7–82.3% fewer denoising NFEs than Vanilla and 1.7–5.8$\times$ end-to-end speedup. Beyond the Vanilla baseline, BlockBatch consistently improves over Fast-dLLM, reducing NFE in all twelve settings by 26.6% on average and up to 33.6% on LLaDA-Instruct-8B/MBPP. It is also faster than or tied with Fast-dLLM in all settings, reaching up to 2.05$\times$

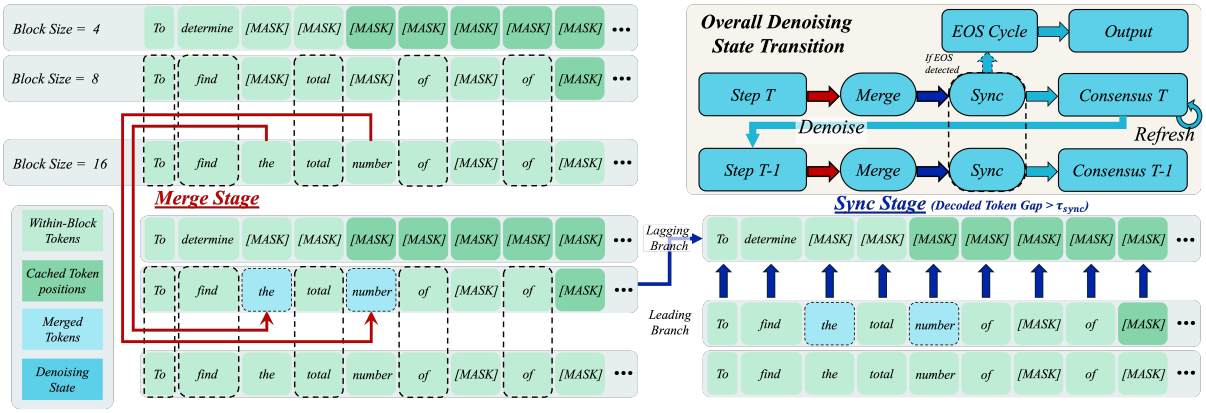


Figure 5: **Overview of BlockBatch.** The left and bottom part illustrates confidence-gated merge and leader-based synchronization. The upper-right panel summarizes the global denoising state transition: each step performs merge, sync, and periodic refresh. If a branch predicts EOS before all preceding positions are decoded, it enters an EOS cycle until the prefix before EOS is continuous, preventing premature termination before final output.

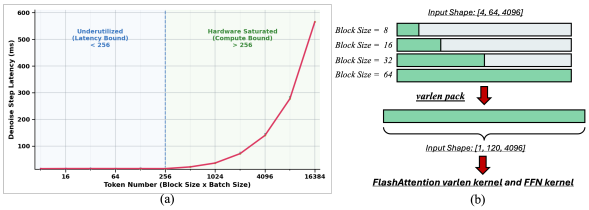


Figure 6: **(a)** Per-step denoise latency on H200 vs. total token count. **(b) BlockBatch Denoise Step.** Unmasked query positions from all branches are packed into one variable-length buffer consumed by both varlen attention and the FFN, while a shared unified KV cache preserves per-branch attention semantics exactly.

additional speedup on LLaDA-1.5-8B/HumanEval. Compared with LocalLeap, BlockBatch obtains lower or tied NFE in all settings.

5.3 Ablation Studies

BlockBatch depends on three main design choices: the synchronization threshold, the refresh interval, and the set of block sizes used for parallel exploration. These parameters control the central trade-off of the method: allowing enough branch diversity to discover efficient trajectories while limiting computational overhead.

Synchronization Threshold. The synchronization threshold determines how far the leading branch is allowed to advance before a lagging branch is synchronized to it (See Appendix A, Tab. 2). This threshold is important because it controls the balance between exploration and stability. If synchronization is triggered too aggressively, branches are forced to follow the current leader too early, falling back to a greedy regime that does not guarantee a globally optimal trajectory.

On the other hand, if the threshold is too large, lagging branches can remain trapped in slow or stale block-denoising regions. In the KV-space view, these branches may continue accumulating local cache drift without contributing useful progress. Synchronization therefore acts as a branch-space contraction mechanism: it preserves useful exploration within a bounded window while preventing slow branches from wasting computation after they have fallen too far behind. The ablation results show that the best performance is obtained at an intermediate threshold, where branch diversity is maintained but unproductive drift is controlled.

Refresh Interval. The refresh interval R controls how often a full-sequence refresh is applied. (See appendix A, table 3) From the refresh contraction bound, stability requires $\beta(1+\lambda_b)^R < 1$, which implies $R < \frac{\log(1/\beta)}{\log(1+\lambda_b)}$. Thus, R cannot be too large: if refresh is too infrequent, accumulated block-denoise error can dominate the contraction effect of full-sequence refresh. However, R also cannot be too small in practice, because frequent full-sequence refreshes increase model-call overhead and end-to-end latency. Therefore, the refresh interval must lie in a feasible window: small enough to control KV-cache drift, but large enough to amortize the cost of global recomputation. The ablation results in Tab. 3 confirm this behavior, showing that the optimal refresh setting is neither overly frequent nor overly sparse.

Table 1: Benchmark results for Vanilla, Fast-dLLM Dual Cache, LocalLeap, and BlockBatch across LLaDA-1.5-8B, LLaDA-Instruct-8B, and Dream-Base-7B. For Fast-dLLM Dual Cache, we report only the block size 32 setting. *Notation:* Fast-dLLM denotes Fast-dLLM Dual Cache with block size 32. Green cells indicate BlockBatch results. Subscripts on Latency show speedup over Vanilla; subscripts on NFE show reduction percentage relative to Vanilla.

		LLaDA-1.5-8B			**LLaDA-Instruct-8B**			**Dream-Base-7B**		
Task	Method	Lat. (s) ↓	NFE ↓	ACC (%) ↑	Lat. (s) ↓	NFE ↓	ACC (%) ↑	Lat. (s) ↓	NFE ↓	ACC (%) ↑
GSM8K	Vanilla	12.5	256.0	79.91	13.1	256.0	77.10	10.4	256.0	**75.13**
	Fast-dLLM	4.9↓2.6×	84.7↓67%	79.68	2.7↓4.9×	86.4↓66%	**78.70**	3.7↓2.8×	162.3↓37%	73.62
	LocalLeap	4.7↓2.7×	72.5↓72%	**81.27**	2.3↓5.7×	64.4↓75%	78.16	6.5↓1.6×	142.9↓44%	72.71
	BlockBatch	4.2↓3.0×	**63.6**↓75%	77.56	2.4↓5.5×	**63.2**↓75%	77.48	3.7↓2.8×	**133.8**↓48%	72.48
MATH	Vanilla	9.04	256.0	**34.78**	9.7	256.0	**40.14**	8.1	256.0	**40.10**
	Fast-dLLM	5.5↓1.6×	104.4↓59%	33.20	3.3↓2.9×	107.8↓58%	37.12	2.5↓3.2×	121.9↓52%	39.34
	LocalLeap	3.6↓2.5×	80.3↓69%	31.74	2.8↓3.5×	85.3↓67%	32.18	3.3↓2.5×	**92.8**↓64%	37.98
	BlockBatch	3.4↓2.7×	**77.6**↓70%	32.24	2.4↓4.0×	**80.4**↓69%	37.20	2.2↓3.7×	**92.8**↓64%	39.44
HumanEval	Vanilla	5.77	256.0	**43.29**	5.7	256.0	**40.24**	4.7	256.0	50.00
	Fast-dLLM	4.3↓1.3×	88.7↓65%	39.02	2.6↓2.2×	90.4↓65%	36.59	3.1↓1.5×	156.1↓39%	**52.44**
	LocalLeap	2.0↓2.9×	75.8↓70%	42.07	2.1↓2.7×	69.8↓73%	35.98	2.6↓1.8×	123.9↓52%	48.78
	BlockBatch	2.1↓2.7×	**63.0**↓75%	39.63	2.0↓2.9×	**64.6**↓75%	38.41	2.8↓1.7×	**112.5**↓56%	**52.44**
MBPP	Vanilla	9.50	256.0	**43.40**	9.9	256.0	**41.40**	8.0	256.0	**55.80**
	Fast-dLLM	4.2↓2.3×	76.1↓70%	38.00	2.2↓4.5×	68.2↓73%	38.40	2.6↓3.1×	111.9↓56%	53.20
	LocalLeap	2.7↓3.5×	58.0↓77%	40.20	1.9↓5.2×	56.1↓78%	**41.40**	2.9↓2.8×	82.8↓68%	54.60
	BlockBatch	2.6↓3.7×	**54.5**↓79%	40.00	1.7↓5.8×	**45.3**↓82%	39.60	2.3↓3.5×	**81.1**↓68%	52.00

Block-Size Configuration. The choice of block sizes determines the diversity of decoding trajectories explored by BlockBatch (Shown in Fig. 7). Using multiple block sizes allows the method to exploit idle GPU capacity by evaluating several denoising schedules in parallel. Smaller blocks provide conservative, locally stable progress, while larger blocks can decode more aggressively when the model is confident. Combining them increases the probability that at least one branch follows an efficient trajectory within the synchronization window.

However, adding more branches is not free. Although more block sizes can reduce NFE by increasing exploration and enabling useful synchronization, each additional branch also increases batching, cache-management, and refresh overhead. Therefore, the best block-size configuration must balance NFE reduction against end-to-end latency. The ablation results in Fig. 7 show that multi-scale block-size sets are more effective than narrow configurations, but also that the benefit saturates once the additional branches no longer provide enough useful trajectory diversity to justify their overhead.

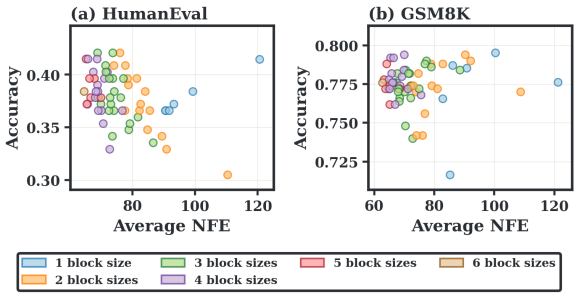


Figure 7: Each point represents one LLaDA BlockBatch candidate block-size set $S \subseteq \mathcal{B}$, where $\mathcal{B} = \{4, 8, 16, 32, 64, 128\}$. Colors encode the set cardinality $|S|$. Thus, the singleton group contains $\binom{6}{1} = 6$ fixed block-size configurations, the two-block-size group contains $\binom{6}{2} = 15$ pairwise configurations, and in general the k-block-size group contains $\binom{6}{k}$ candidate sets.

6 Conclusion

We presented BlockBatch, a training-free framework that treats block size as a branching axis for dLLM inference. By coordinating multiple block-size branches through confidence-gated merging, leader-based synchronization, and periodic full-sequence refresh within a single fused forward pass, BlockBatch consistently reduces NFEs and latency on LLaDA and Dream across four benchmarks while preserving accuracy over state-of-the-art dLLM inference frameworks. Block-size diversity is a practical, underexplored axis for branch-parallel diffusion decoding.

7 Limitations

BlockBatch is designed as a training-free inference-time acceleration method, and its effectiveness depends on a decoding regime in which block-size diversity produces compatible but non-identical trajectories. In low-diversity regimes, such as near-deterministic or extremely easy prompts where all selected block sizes converge to the same decoded sequence with little or no bifurcation, additional branches contribute limited new information while still introducing coordination overhead. Conversely, BlockBatch also assumes that the selected block sizes are individually stable enough to provide useful candidate trajectories. If a model becomes unreliable under large block sizes, such branches may reduce accuracy and should be adaptively deselected during inference rather than retained throughout the full decoding process.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*.

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

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024a. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235. PMLR.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024b. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *Preprint*, arXiv:2406.02069.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. 2022. A continuous time framework for discrete denoising models. *Preprint*, arXiv:2205.14987.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. 2024. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *Preprint*, arXiv:2402.04997.

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, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. 2023. Analog bits: Generating discrete data using diffusion models with self-conditioning. *Preprint*, arXiv:2208.04202.

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

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. 2021a. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*.

Hengyu Fu, Baihe Huang, Virginia Adams, Charles Wang, Venkat Srinivasan, and Jiantao Jiao. 2025. From bits to rounds: Parallel decoding with exploration for diffusion language models.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025. Scaling diffusion language models via adaptation from autoregressive models. *Preprint*, arXiv:2410.17891.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021a. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. *NeurIPS*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020a. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020b. Denoising diffusion probabilistic models. *Preprint*, arXiv:2006.11239.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forr
'e, and Max Welling. 2021a. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems*.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021b. Argmax flows and multinomial diffusion: Learning categorical distributions. *Preprint*, arXiv:2102.05379.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S. Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. 2025. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *Preprint*, arXiv:2505.21467.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. 2026. d^2Cache: Accelerating diffusion-based LLMs via dual adaptive caching. In *International Conference on Learning Representations*. Poster.

Fanheng Kong, Jingyuan Zhang, Yahui Liu, Zirui Wu, Yu Tian, and Guorui Zhou. 2025a. Accelerating diffusion llm inference via local determinism propagation.

Fanheng Kong, Jingyuan Zhang, Yahui Liu, Zirui Wu, Yu Tian, Guorui Zhou, and 1 others. 2025b. Accelerating diffusion llm inference via local determinism propagation. *arXiv preprint arXiv:2510.07081*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. Infinigen: Efficient generative inference of large language models with dynamic kv cache management. *Preprint*, arXiv:2406.19707.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Preprint*, arXiv:2404.14469.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025. dLLM-Cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Preprint*, arXiv:2305.17118.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. dKV-Cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems*. Poster.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2025a. Scaling up masked diffusion models on text.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025b. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. 2025. Diffusion beats autoregressive in data-constrained settings. *arXiv preprint arXiv:2507.15857*.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024a. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024b. Simple and effective masked diffusion language models. *Preprint*, arXiv:2406.07524.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Mariano Marroquin, Justin T. Chiu, Alexander M. Rush, and Volodymyr Kuleshov. 2024c. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 130136–130184.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *Preprint*, arXiv:1911.02150.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015a. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015b. Deep unsupervised learning using nonequilibrium thermodynamics. *Preprint*, arXiv:1503.03585.

Jiaming Song, Chenlin Meng, and Stefano Ermon. 2021a. Denoising diffusion implicit models. In *International Conference on Learning Representations*.

Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021b. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *Preprint*, arXiv:2406.10774.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. 2023. Digress: Discrete denoising diffusion for graph generation. *Preprint*, arXiv:2209.14734.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. 2025a. Fast-dllm v2: Efficient block-diffusion llm. *Preprint*, arXiv:2509.26328.

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

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2026. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *International Conference on Learning Representations*. Poster.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*.

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

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

Siyue Zhang, Yilun Zhao, Liyuan Geng, Arman Cohan, Anh Tuan Luu, and Chen Zhao. 2025. Diffusion vs. autoregressive language models: A text embedding perspective. *Preprint*, arXiv:2505.15045.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*.

Appendix

- Sec. A: Threshold Ablations
- Sec. B: Token level characterization
  - Later-Stage Consensus Token Analysis
  - Token Bifurcation Case studies
- Sec. C: KV level characterization: proofs and more case studies
  - Basic Operations
  - Proposition 1: Block Denoise vs. Full Refresh
  - Proposition 2: Periodic Refresh Stabilizes Block Denoising
  - Proposition 3: Branch Bifurcation as Tangent-Space Separation
  - Case Study: KV Cache Vector Space Properties
- Sec. D: Block Batching Algorithm
  - Fused Block Batching with a Unified KV Cache
  - Merge and Synchronization Policy
- Sec. E: Reproducibility, Artifacts and Computer Details
  - Scientific Artifacts
  - Artifact Licenses and Intended Use
  - Evaluation Setup
  - Computational Infrastructure and Budget

A Ablation Tables

Table 2: Sync-threshold τ ablation. **Bold** indicates the best value within each column.

	LLaDA 8B-Inst.				Dream Base-7B			
	GSM8K		HumanEval		GSM8K		HumanEval	
τ	Acc.	NFE	Acc.	NFE	Acc.	NFE	Acc.	NFE
4	0.773	**67.6**	0.354	69.6	0.726	**138.2**	**0.494**	122.7
8	0.770	68.6	0.378	69.9	**0.732**	138.7	**0.494**	120.5
16	**0.784**	69.9	0.384	**69.4**	0.725	139.4	0.470	**114.6**
32	0.780	70.6	**0.415**	70.5	0.719	140.6	0.476	117.3
64	0.780	73.1	0.384	71.6	0.703	143.9	0.427	116.7

Table 3: Refresh-interval R ablation. **Bold** indicates the best value within each column.

	LLaDA 8B-Inst.				Dream Base-7B			
	GSM8K		HumanEval		GSM8K		HumanEval	
R	Acc.	NFE	Acc.	NFE	Acc.	NFE	Acc.	NFE
4	0.778	**59.0**	0.372	**56.8**	**0.780**	143.0	**0.543**	120.8
8	0.784	59.9	0.378	58.9	**0.780**	138.9	0.512	**110.1**
16	0.770	61.1	**0.402**	62.4	0.752	137.5	0.476	111.8
32	0.776	62.7	0.384	64.6	0.730	134.7	0.524	112.5
64	**0.788**	64.6	0.348	67.1	0.744	**134.2**	0.494	112.8
128	0.752	65.6	0.354	71.2	0.746	135.2	0.500	117.2

B Appendix: Token-Level Characterization

B.1 Later-Stage Consensus and Bifurcation Tokens

We further analyze token-level branch behavior by separating two phenomena: *later-stage consensus*, where previously diverged branches produce the same token again at a later position, and *bifurcation*, where two block-size trajectories first commit different tokens after a shared prefix. These two phenomena play different roles. Later-stage consensus indicates local token agreement, but does not imply that the full trajectories have re-converged. Bifurcation tokens, by contrast, mark the positions at which block-size branches begin to follow distinct conditional generation paths.

Fig. 8 and Fig. 9 characterize generated token positions by linguistic category—digit, operator, whitespace, word, and other—and by cross-block-size agreement. Each stacked bar reports the average number of positions per sequence in that category, while the shading indicates how many of the six block-size configurations produced the same token. The overlaid curve shows the mean agreement score for each category.

A clear domain-specific pattern appears. In HumanEval (Fig. 8), whitespace and operator tokens achieve the highest agreement and are dominated by $6/6$ agreement, reflecting the syntactic regularity of code generation. In GSM8K (Fig. 9), digit tokens are more frequent (avg. 61.2 positions per sequence, compared with 23.7 in HumanEval) and also show high agreement, consistent with the repeated numerical structure of arithmetic reasoning traces. Across both domains, the highest-agreement categories are mostly low-semantic-weight tokens: they control formatting, syntax, or local surface structure rather than the main reasoning content.

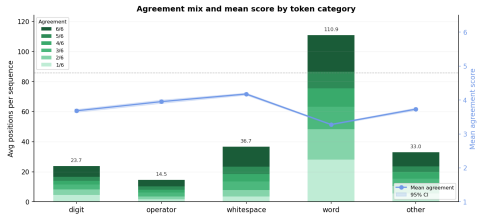


Figure 8: **Token-category agreement profile for HumanEval.** Each stacked bar shows the average number of generated-token positions in a category, shaded by the fraction of block-size configurations that produce the same token at that position ($6/6$ = darkest green; $1/6$ = lightest). The blue curve gives the mean agreement score per category. Whitespace and operator tokens dominate later-stage consensus, reflecting the syntactic regularity of code generation.

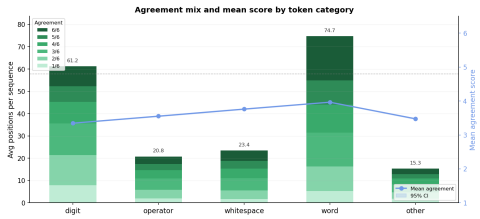


Figure 9: **Token-category agreement profile for GSM8K.** Digit tokens are substantially more prevalent than in HumanEval (avg. 61.2 positions per sequence) and exhibit high mean agreement, consistent with the repetitive numerical structure of arithmetic reasoning chains. Word tokens are the most frequent category overall (avg. 74.7 positions per sequence) but show lower per-position agreement, indicating that they more commonly mark bifurcation points where block-size trajectories diverge.

This category-level pattern explains why later-stage consensus is useful but limited. Although branches may re-agree on individual tokens after an earlier split, those tokens are often formatting or locally constrained tokens rather than semantically decisive reasoning steps. To test whether such tokens can meaningfully steer generation, we perform a seeded-consensus experiment. For each

sample, we identify token positions with strict $6/6$ cross-block-size agreement after the initial shared prefix and pre-fill those positions before normal denoising begins.

Table 4: **Effect of seeding later-stage consensus tokens on Dream-v0-Instruct-7B.** *Seeded* pre-fills later-stage consensus positions before denoising, where consensus is defined as $6/6$ cross-block-size agreement after the initial shared prefix. On average, this seeds 4.6 tokens per GSM8K sample and 3.5 tokens per HumanEval sample. ΔAcc. is Seeded minus Baseline; ΔNFE is Baseline minus Seeded, so positive ΔNFE indicates reduced denoising cost.

Task	Block size	Accuracy (%)			Average NFE		
		Baseline	Seeded	ΔAcc.	Baseline	Seeded	ΔNFE
GSM8K	4	76.04	74.75	-1.29	154.2	152.9	1.3
GSM8K	8	76.04	74.91	-1.13	104.3	102.3	2.0
GSM8K	16	74.98	73.62	-1.36	80.0	77.5	2.5
GSM8K	32	73.69	73.24	-0.45	67.9	65.4	2.5
GSM8K	64	70.58	70.13	-0.45	64.8	61.6	3.2
GSM8K	128	41.77	42.61	$+0.84$	81.8	79.0	2.8
HumanEval	4	54.88	50.00	-4.88	174.3	173.7	0.6
HumanEval	8	55.49	54.27	-1.22	129.0	125.3	3.7
HumanEval	16	55.49	55.49	$+0.00$	102.6	100.1	2.5
HumanEval	32	55.49	56.71	$+1.22$	86.4	85.8	0.6
HumanEval	64	53.05	52.44	-0.61	82.7	81.9	0.8
HumanEval	128	33.54	34.15	$+0.61$	67.0	64.1	2.9

Table 4 shows that seeding later-stage consensus tokens produces only small NFE reductions and does not systematically improve accuracy. On average, only 4.6 GSM8K tokens and 3.5 HumanEval tokens per sample satisfy the strict post-prefix $6/6$ agreement criterion. This supports the view that later-stage consensus tokens are locally reusable but usually not strong enough to determine the global denoising trajectory. Therefore, the semantically important variation between branches is better captured by where they first bifurcate, rather than by the small number of tokens on which they later re-agree.

We therefore measure the average bifurcation length between every pair of single-block-size generations. For a pair of block sizes, bifurcation length is the length of the common generated-token prefix before the two outputs first differ. Smaller values indicate earlier trajectory separation.

Table 5: **Average bifurcation length between single-block-size generations.** For each pair of block sizes, bifurcation length is defined as the length of the common generated-token prefix before the two outputs first differ, following `analyze_bifurcation.py`. When the two outputs have different lengths, the shorter generated sequence is used. Lower values indicate earlier trajectory divergence. Bold entries denote the earliest bifurcation within each model–dataset row.

Model	Dataset	Block-size pair														
		4–8	4–16	4–32	4–64	4–128	8–16	8–32	8–64	8–128	16–32	16–64	16–128	32–64	32–128	64–128
Dream	GSM8K	23.4	18.0	16.0	15.9	**15.6**	25.5	20.4	19.0	18.5	27.1	23.3	22.1	32.1	29.8	44.3
Dream	HumanEval	34.9	22.0	17.9	17.8	**16.4**	37.1	20.9	22.6	19.0	28.1	23.3	23.1	34.5	29.6	47.2
Dream	MATH	32.6	23.4	22.0	**21.8**	22.0	34.1	27.5	26.5	26.2	40.3	34.5	33.4	47.5	43.1	59.9
Dream	MBPP	33.3	21.6	**20.9**	21.7	21.7	32.6	27.0	25.6	26.9	36.7	33.1	29.0	53.8	43.5	56.7
LLaDA	GSM8K	35.2	24.8	22.0	21.2	**20.6**	32.7	26.4	25.0	23.4	34.2	28.0	25.8	38.9	33.0	48.7
LLaDA	HumanEval	64.5	48.5	42.6	31.3	32.5	52.2	39.0	**30.0**	30.5	53.7	38.0	37.4	52.6	44.0	54.3
LLaDA	MATH	34.0	24.7	22.4	22.2	**22.2**	34.5	28.0	25.4	25.5	38.0	30.6	29.7	43.4	39.0	54.7
LLaDA	MBPP	31.6	21.9	**17.9**	19.1	18.9	28.3	20.8	19.5	21.1	26.7	22.3	23.3	26.2	25.5	37.7

Table 5 shows that substantially different block-size pairs generally bifurcate earlier than nearby block-size pairs. For example, pairs involving small and large blocks, such as 4–128 or 8–128, often split earlier than adjacent pairs such as 32–64 or 64–128. This pattern suggests that block-size diversity is not merely redundant decoding. Different block sizes expose distinct token-level trajectories, while still often preserving enough shared structure to support synchronization and selective merging.

The exact bifurcation position depends on both the dataset and the model. Code-generation tasks often share long syntactic prefixes before diverging in implementation details, while arithmetic reasoning tasks can bifurcate through small changes in wording, intermediate computation, or final-answer formatting. Table 6, 7, 8 provides case studies using raw output spans. Rather than summarizing each generation abstractly, the table shows the first visible split after the shared reasoning prefix and the final answer span for each inference regime.

B.2 Token Bifurcation Case studies

This subsection reports three `GSM8K` examples from `LLaDA-8B-Instruct`. For each example, we show the first raw output bifurcation among the six single block-size branches and the terminal answer span generated by each method. The bifurcation position is computed after whitespace normalization for readability. The reported continuation is the raw branch-local text beginning at the first divergent token.

`GSM8K` sample 1. Target answer: 18. The six single block-size branches share the prefix … `total number of eggs laid` and first bifurcate at token position 26.

Table 6: **Token bifurcation on GSM8K sample 1.** Several branches compute the correct value 18, but strict exact match depends on whether the terminal span contains the required #### answer marker.

Method	First divergent continuation	Terminal answer span	Pred.	EM	NFE
Vanilla	by Janet's ducks per day	farmers' market. #### 18	18	Y	256
BS=4	by Janet's ducks per day	The final answer is: \boxed{18}	invalid	N	154
BS=8	by the ducks per day	farmers' market. #### 18	18	Y	120
BS=16	by Janet's ducks per day	farmers' market. #### 18	18	Y	121
BS=32	by Janet's ducks per day	The final answer is: \boxed{18}	invalid	N	123
BS=64	per day. 2. Subtract	farmers' market. #### 18	18	Y	111
BS=128	by Janet's ducks per day	Janet makes $18 every day at the farmers' market.	invalid	N	105
BlockBatch	by Janet's ducks per day	farmers' market. #### 24	24	N	557

GSM8K sample 2. Target answer: 3. The six single block-size branches share the prefix ... `robe takes 2 bolts of blue fiber and half that` and first bifurcate at token position 11.

Table 7: **Token bifurcation on GSM8K sample 2.** Most branches preserve the correct arithmetic trajectory. Failures are mainly strict-format failures caused by missing terminal #### extraction.

Method	First divergent continuation	Terminal answer span	Pred.	EM	NFE
Vanilla	much white fiber. To find	total of 3 bolts. #### 3	3	Y	256
BS=4	much white fiber. First, calculate	it takes is 3. #### 3	3	Y	110
BS=8	much white fiber. So, the	= 3 \text{ bolts} \] #### 3	3	Y	78
BS=16	amount of white fiber. So,	= 3 \text{ bolts} \] #### 3	3	Y	66
BS=32	amount of white fiber. First,	robe takes a total of 3 bolts.	invalid	N	76
BS=64	amount of white fiber. First,	2 bolts + 1 bolt = 3 bolts. #### 3	3	Y	87
BS=128	amount of white fiber.	blue fiber and half that amount of white fiber.	invalid	N	77
BlockBatch	amount of white fiber. Half	white fiber = 3 bolts. #### 3	3	Y	192

GSM8K sample 3. Target answer: 70,000. The six single block-size branches share the prefix ... `need to follow these steps: 1. Calculate the` and first bifurcate at token position 13.

Table 8: **Token bifurcation on GSM8K sample 3.** This example shows a semantic bifurcation. Branches that continue with `total cost` are more likely to preserve the repair-cost term, while branches that continue with `new value` often omit it and terminate near 120,000.

Method	First divergent continuation	Terminal answer span	Pred.	EM	NFE
Vanilla	new value of the house	Therefore, Josh's profit is $120,000.	invalid	N	256
BS=4	total cost of the house	incurred a loss. #### -10,000	−10,000	N	133
BS=8	total cost of the house	Josh's profit is $70,000. #### 70000	70000	Y	96
BS=16	new value of the house	Josh made a profit of $120,000.	invalid	N	106
BS=32	new value of the house	Josh's profit is $120,000.	invalid	N	96
BS=64	new value of the house	Josh made a profit of $10,000.	invalid	N	123
BS=128	new value of the house	Therefore, Josh's profit is $120,000.	invalid	N	101
BlockBatch	total cost of the house	Josh made a profit of $100,000.	invalid	N	330

Takeaway. Across the three examples, the first bifurcation occurs early: token positions 26, 11, and 13 after whitespace normalization. Samples 1 and 2 show mostly formatting bifurcation: several branches compute the correct terminal number but fail strict exact match because they emit `\boxed{}` or omit the `####` marker. Sample 3 shows reasoning bifurcation: the first divergent continuation already separates branches that track total cost from branches that focus on new value, and this early split determines whether the repair cost is preserved in the terminal answer. Therefore, raw token bifurcation is useful for distinguishing extraction artifacts from genuine reasoning drift. BlockBatch benefits from branch diversity only when the selected terminal branch preserves the correct arithmetic trajectory and emits an extractor-compatible final answer.

C Appendix: KV Cache Vector Space Formulation

C.1 Basic Operations

We model each branch cache as a fixed vectorization of all key and value tensors across layers, heads, and sequence positions. Thus, at event step t, branch b has

$$K_t^{(b)} \in \mathbb{R}^D.$$

All norms below are Euclidean norms on this fixed vectorization. The event index t ranges over block-

denoise and full-refresh events.

Let

$$x_t^{(b)} \in \widetilde{\mathcal{V}}^L$$

be the current token state, where $\widetilde{\mathcal{V}}$ contains the vocabulary and the mask token. Let

$$F(x_t^{(b)}) \in \mathbb{R}^D$$

denote the full-sequence KV-cache map obtained by recomputing the cache from the complete current sequence.

We distinguish two cache-update operations.

1. **Full-sequence refresh.** A full refresh recomputes the cache from the complete current token state:

$$K_{t+1}^{(b)} = F(x_t^{(b)}).$$

The corresponding cache correction is

$$\Delta_{\text{full},t}^{(b)} = F(x_t^{(b)}) - K_t^{(b)}.$$

2. **Block denoise.** A block-denoise step updates an active block $B_t^{(b)} \subseteq \{1, \ldots, L\}$, with $|B_t^{(b)}| = s_b$, while reusing cached context outside the block:

$$K_{t+1}^{(b)} = K_t^{(b)} + \Delta_{\text{blk},t}^{(b)}.$$

This appendix analyzes how the active block size affects the expected magnitude of the local KV correction. Synchronization and branch selection are policy operations in BlockBatch and are not needed for the following KV-norm bounds.

C.2 Proposition 1: Expected Block-Local KV Movement

The following result formalizes the average-case locality of block denoising. The $\sqrt{m/L}$ factor does not hold for every fixed block and every fixed vector. It holds in expectation when a block of size m covers an m/L fraction of positions on average.

Positional KV-energy notation. For a flattened KV-cache perturbation $v \in \mathbb{R}^D$, write

$$v = (v_1, \ldots, v_L),$$

where v_p contains all key/value coordinates associated with sequence position p, across all layers, heads, and key/value channels. For a block $B \subseteq \{1, \ldots, L\}$, let P_B be the orthogonal projection that keeps coordinates associated with positions in B and zeros all other coordinates. Then

$$\|P_B v\|_2^2 = \sum_{p \in B} \|v_p\|_2^2.$$

Here "energy" means squared Euclidean norm of the KV perturbation.

Lemma 1 (Expected block-projection energy)**.** *Let B be a random block of size m such that every position has the same inclusion probability:*

$$\Pr(p \in B) = \frac{m}{L}, \qquad p = 1, \ldots, L.$$

Then, for any fixed KV perturbation $v \in \mathbb{R}^D$,

$$\mathbb{E}_B\left[\|P_B v\|_2^2\right] = \frac{m}{L}\|v\|_2^2.$$

Consequently,

$$\mathbb{E}_B\left[\|P_B v\|_2\right] \le \sqrt{\frac{m}{L}}\|v\|_2.$$

Proof. By definition,

$$\|P_B v\|_2^2 = \sum_{p=1}^{L} \mathbf{1}\{p \in B\}\|v_p\|_2^2.$$

Taking expectation over B,

$$\begin{aligned}
\mathbb{E}_B\left[\|P_B v\|_2^2\right] &= \sum_{p=1}^{L} \mathbb{E}_B[\mathbf{1}\{p \in B\}]\|v_p\|_2^2 \\
&= \sum_{p=1}^{L} \Pr(p \in B)\|v_p\|_2^2 \\
&= \frac{m}{L}\sum_{p=1}^{L} \|v_p\|_2^2 \\
&= \frac{m}{L}\|v\|_2^2.
\end{aligned}$$

By Jensen's inequality,

$$\mathbb{E}_B[\|P_B v\|_2] \le \sqrt{\mathbb{E}_B[\|P_B v\|_2^2]} = \sqrt{\frac{m}{L}}\|v\|_2.$$

□

Proposition 1 (Expected block-local KV update bound)**.** *Let $m = |B_t^{(b)}|$ and let L be the sequence length. Consider the full-refresh correction*

$$\Delta_{\text{full},t}^{(b)} = F(x_t^{(b)}) - K_t^{(b)}.$$

Decompose the block-denoise correction as

$$\Delta^{(b)}_{\mathrm{blk},t} = P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t} + r^{(b)}_t.$$

The residual $r^{(b)}_t$ is defined by this equation and captures the part of the block-denoise correction that is not explained by the block-restricted full-refresh correction.

If the active block satisfies

$$\Pr(p \in B^{(b)}_t) = \frac{m}{L}, \qquad p = 1, \dots, L,$$

then

$$\mathbb{E}_B\left[\left\|P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t}\right\|_2^2\right] = \frac{m}{L}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2^2.$$

Therefore,

$$\mathbb{E}_B\left[\left\|P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t}\right\|_2\right] \le \sqrt{\frac{m}{L}}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2.$$

With the residual included,

$$\mathbb{E}_B\left[\left\|\Delta^{(b)}_{\mathrm{blk},t}\right\|_2\right] \le \sqrt{\frac{m}{L}}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2 + \mathbb{E}_B\left[\left\|r^{(b)}_t\right\|_2\right].$$

Proof. Apply the expected block-projection lemma to $v = \Delta^{(b)}_{\mathrm{full},t}$:

$$\mathbb{E}_B\left[\left\|P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t}\right\|_2^2\right] = \frac{m}{L}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2^2.$$

Jensen's inequality gives

$$\mathbb{E}_B\left[\left\|P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t}\right\|_2\right] \le \sqrt{\frac{m}{L}}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2.$$

Using

$$\Delta^{(b)}_{\mathrm{blk},t} = P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t} + r^{(b)}_t$$

and the triangle inequality,

$$\left\|\Delta^{(b)}_{\mathrm{blk},t}\right\|_2 \le \left\|P_{B^{(b)}_t}\Delta^{(b)}_{\mathrm{full},t}\right\|_2 + \left\|r^{(b)}_t\right\|_2.$$

Taking expectation over B yields

$$\mathbb{E}_B\left[\left\|\Delta^{(b)}_{\mathrm{blk},t}\right\|_2\right] \le \sqrt{\frac{m}{L}}\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2 + \mathbb{E}_B\left[\left\|r^{(b)}_t\right\|_2\right].$$

□

Comparison with full refresh. A full refresh applies the complete correction

$$\Delta^{(b)}_{\mathrm{full},t} = F(x^{(b)}_t) - K^{(b)}_t.$$

A block-denoise step applies only the active-block component of this correction, up to residual effects. Therefore, if

$$\mathbb{E}_B\left[\left\|r^{(b)}_t\right\|_2\right] < \left(1 - \sqrt{\frac{m}{L}}\right)\left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2,$$

then

$$\mathbb{E}_B\left[\left\|\Delta^{(b)}_{\mathrm{blk},t}\right\|_2\right] < \left\|\Delta^{(b)}_{\mathrm{full},t}\right\|_2.$$

Interpretation. This proposition uses one averaging condition: each token position is included in an active block with probability m/L. Under that condition, the direct block-local component contains an m/L fraction of squared KV-update energy in expectation. The norm therefore scales by $\sqrt{m/L}$. The residual $r^{(b)}_t$ covers effects not captured by a pure projection of the full-refresh correction, such as attention spillover outside the active block. Thus the result gives an average-case explanation for why block-denoise events produce smaller KV movements than full-sequence refreshes, without requiring extra concentration, smoothness, or interpolation assumptions.

C.3 Proposition 2: Periodic Refresh Stabilizes Block Denoising

Let

$$F : \widetilde{\mathcal{V}}^L \to \mathbb{R}^D$$

denote the full-sequence KV-cache map on discrete token states. The reachable full-refresh cache set is

$$\mathcal{C} = \{F(x) : x \in \widetilde{\mathcal{V}}^L\}.$$

We call $\mathcal{C}$ the reachable full-refresh cache set rather than a smooth manifold, since $\widetilde{\mathcal{V}}^L$ is discrete.

For branch b, define the ideal cache associated with its current token state:

$$K^{(b)}_{\star,t} = F(x^{(b)}_t),$$

and define the cache-consistency error:

$$E^{(b)}_t = \left\|K^{(b)}_t - K^{(b)}_{\star,t}\right\|_2 = \left\|K^{(b)}_t - F(x^{(b)}_t)\right\|_2.$$

This measures the deviation between the maintained approximate cache and the cache obtained by a full forward pass on the same token state.

Proposition 2 (Refresh-stabilized cache error under bounded drift)**.** *Suppose each block-denoise step satisfies*

$$E^{(b)}_{t+1} \le (1+\lambda_b)E^{(b)}_t + \epsilon_B, \qquad \lambda_b \ge 0, \quad \epsilon_B \ge 0,$$

and each full refresh satisfies

$$E^{(b)}_{t+1} \le \beta E^{(b)}_t + \epsilon_F, \qquad 0 \le \beta < 1, \quad \epsilon_F \ge 0.$$

For an exact full refresh with unchanged token state and exact arithmetic, this corresponds to the special case $\beta = 0$ and $\epsilon_F = 0$.

Suppose one full refresh is applied after every R block-denoise steps. Let t_n denote the event index

immediately after the n-th full refresh and define $Y_n = E^{(b)}_{t_n}$. Let

$$a_b = 1 + \lambda_b, \qquad \rho = \beta a_b^R,$$

and

$$c = \beta \epsilon_B \sum_{i=0}^{R-1} a_b^i + \epsilon_F.$$

If

$$\rho = \beta(1+\lambda_b)^R < 1,$$

then the refresh-boundary errors satisfy

$$Y_n \le \rho^n Y_0 + c\frac{1-\rho^n}{1-\rho}.$$

Consequently,

$$\limsup_{n\to\infty} Y_n \le \frac{c}{1-\rho}.$$

Furthermore, for any within-cycle step $0 \le r < R$,

$$E^{(b)}_{t_n+r} \le a_b^r Y_n + \epsilon_B \sum_{i=0}^{r-1} a_b^i.$$

Proof. Starting immediately after a refresh at time t_n, after R block-denoise steps,

$$E^{(b)}_{t_n+R} \le a_b^R E^{(b)}_{t_n} + \epsilon_B \sum_{i=0}^{R-1} a_b^i.$$

Applying one refresh gives

$$\begin{aligned} E^{(b)}_{t_{n+1}} &\le \beta E^{(b)}_{t_n+R} + \epsilon_F \\ &\le \beta a_b^R E^{(b)}_{t_n} + \beta\epsilon_B \sum_{i=0}^{R-1} a_b^i + \epsilon_F. \end{aligned}$$

Thus,

$$Y_{n+1} \le \rho Y_n + c.$$

Iterating the scalar recurrence yields

$$Y_n \le \rho^n Y_0 + c\sum_{j=0}^{n-1}\rho^j = \rho^n Y_0 + c\frac{1-\rho^n}{1-\rho}.$$

Since $\rho < 1$, taking $n \to \infty$ gives

$$\limsup_{n\to\infty} Y_n \le \frac{c}{1-\rho}.$$

The within-cycle bound follows by applying the block-denoise recurrence for r steps starting from Y_n. □

Interpretation. This result is a stability lemma conditional on bounded block drift and refresh contraction. It states a sufficient condition under which periodic full refreshes prevent cache-consistency error from growing without bound.

C.4 Proposition 3: Projected Cache Separation Diagnostics

We now define the projected coordinate system used in the KV-cache diagnostics. Let $c_0 \in \mathbb{R}^D$ be a fixed nonzero reference cache vector, such as the shared prefill cache or the mean initial branch cache. Define

$$u_0 = \frac{c_0}{\|c_0\|_2}.$$

Let $e_1, e_2 \in u_0^\perp$ be orthonormal tangent directions. In practice, e_1, e_2 may be chosen by PCA, a deterministic sketch, or another fixed branch-independent projection rule. Let

$$U = [u_0, e_1, e_2] \in \mathbb{R}^{D\times 3}.$$

For each branch cache, decompose

$$K_t^{(b)} - c_0 = z_t^{(b)} u_0 + a_t^{(b)} e_1 + q_t^{(b)} e_2 + h_t^{(b)}. \tag{3}$$

where

$$\begin{aligned} z_t^{(b)} &= \langle K_t^{(b)} - c_0, u_0\rangle, \quad a_t^{(b)} = \langle K_t^{(b)} - c_0, e_1\rangle, \\ q_t^{(b)} &= \langle K_t^{(b)} - c_0, e_2\rangle, \quad h_t^{(b)} = (I - UU^\top)(K_t^{(b)} - c_0). \end{aligned} \tag{4}$$

By construction,

$$h_t^{(b)} \perp \mathrm{span}\{u_0, e_1, e_2\}. \tag{5}$$

For two branches i, j, define the full cache distance

$$d^2_{ij,t} = \left\| K_t^{(i)} - K_t^{(j)} \right\|_2^2$$

and the projected cache distance

$$\begin{aligned} d^2_{\mathrm{proj},ij,t} = &\left| z_t^{(i)} - z_t^{(j)} \right|^2 + \left| a_t^{(i)} - a_t^{(j)} \right|^2 \\ &+ \left| q_t^{(i)} - q_t^{(j)} \right|^2. \end{aligned} \tag{6}$$

Proposition 3 (Orthogonal cache-distance decomposition)**.** *Under the orthogonal decomposition above,*

$$\begin{aligned} \left\| K_t^{(i)} - K_t^{(j)} \right\|_2^2 = &\left| z_t^{(i)} - z_t^{(j)} \right|^2 + \left| a_t^{(i)} - a_t^{(j)} \right|^2 \\ &+ \left| q_t^{(i)} - q_t^{(j)} \right|^2 + \left\| h_t^{(i)} - h_t^{(j)} \right\|_2^2. \end{aligned}$$

Therefore,

$$d_{\mathrm{proj},ij,t} \le d_{ij,t}.$$

Proof. The vectors u_0, e_1, e_2 are orthonormal and each residual $h_t^{(b)}$ lies in the orthogonal complement of their span. The result therefore follows from the Pythagorean theorem. □

We define average branch dispersion as

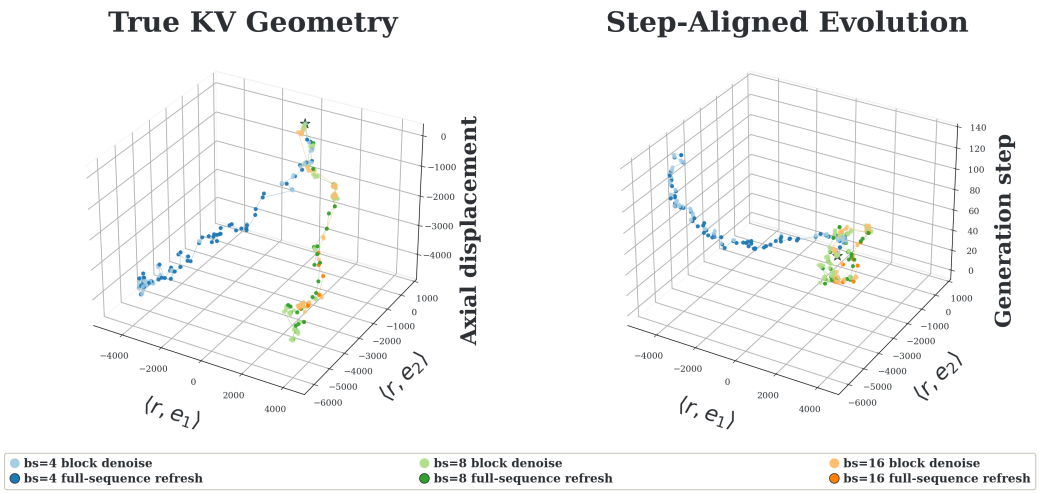
$$D_t = \frac{1}{B(B-1)} \sum_{i \neq j} \left\| K_t^{(i)} - K_t^{(j)} \right\|_2^2 .$$

The projected coordinates (z, a, q) provide a low-dimensional view of one component of the full branch dispersion. Therefore, separation in the plotted (e_1, e_2, z) coordinates is a diagnostic for KV-cache trajectory divergence, not a proof by itself that token predictions must differ.

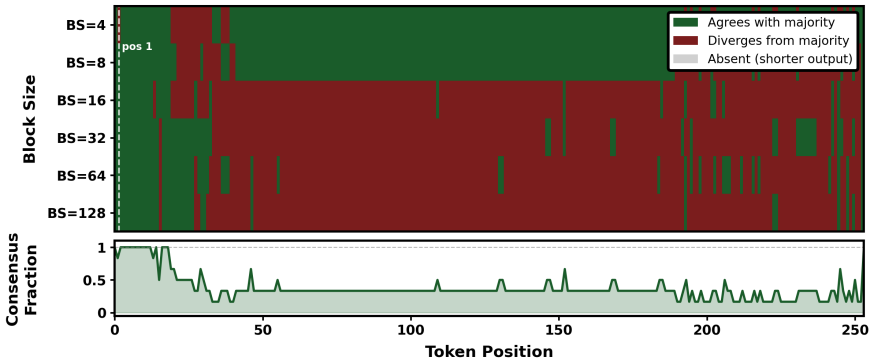
C.5 Case Study: KV Cache Vector Space Properties

To examine whether the geometric claims of Sec. C hold across prompts, we present three `HumanEval` case studies: samples 2, 15, and 19. For each sample, we show two paired diagnostics: (i) the projected KV-cache trajectory in the (e_1, e_2, z) coordinate system of Sec. C.1, where the axial coordinate is $z = \langle K - c_0, u_0 \rangle$ with $u_0 = c_0 / \|c_0\|_2$; and (ii) the token-level consensus heatmap across the six block-size branches that share the prompt and prefill cache. The KV diagnostic reflects geometric behavior of the cache vector, while the consensus heatmap exposes the token-level outcome of that geometry at decoded positions.

Across the three prompts, we observe the same three phenomena predicted by Props. C.2–C.4: adjacent block-denoise events remain confined to small tangent-plane neighborhoods; every large axial excursion in z is initiated by a full-sequence refresh rather than by local denoising; and branches that share a common prefix fan out in (e_1, e_2) at the same generation step where the consensus heatmap records the first divergent token. This consistency suggests that the conic-manifold view of KV-cache evolution is not an artifact of a single prompt and that token-level bifurcation can be read off the KV geometry directly.

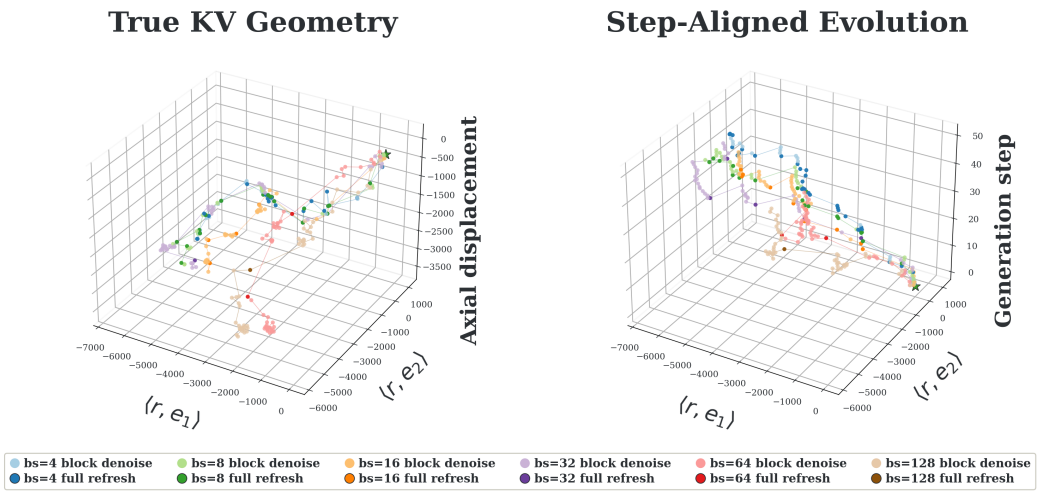


(a) KV-cache trajectory: true KV geometry and step-aligned evolution.

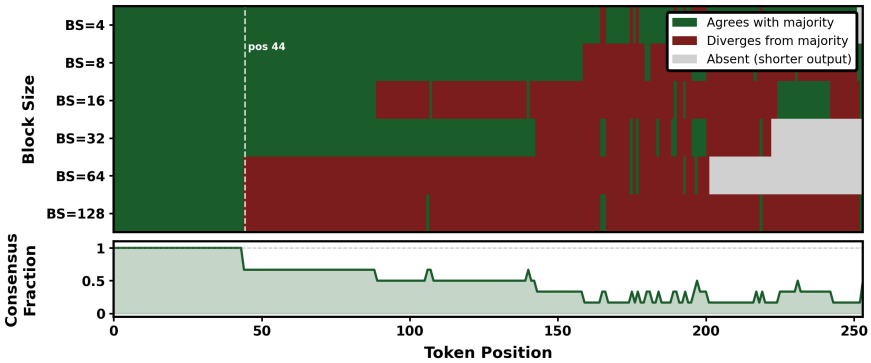


(b) Block-size branch consensus heatmap.

Figure 10: Case study: `HumanEval` sample 2. States from all six block-size branches concentrate along a low-dimensional locus dominated by u_0. Block-denoise events form dense tangent clusters, whereas large axial jumps are initiated only by full-sequence refreshes (Prop. C.2). The consensus heatmap shows that all branches share an identical prefix and then bifurcate at the dashed line, matching the tangent-plane separation visible in the KV trajectory (Prop. C.4).

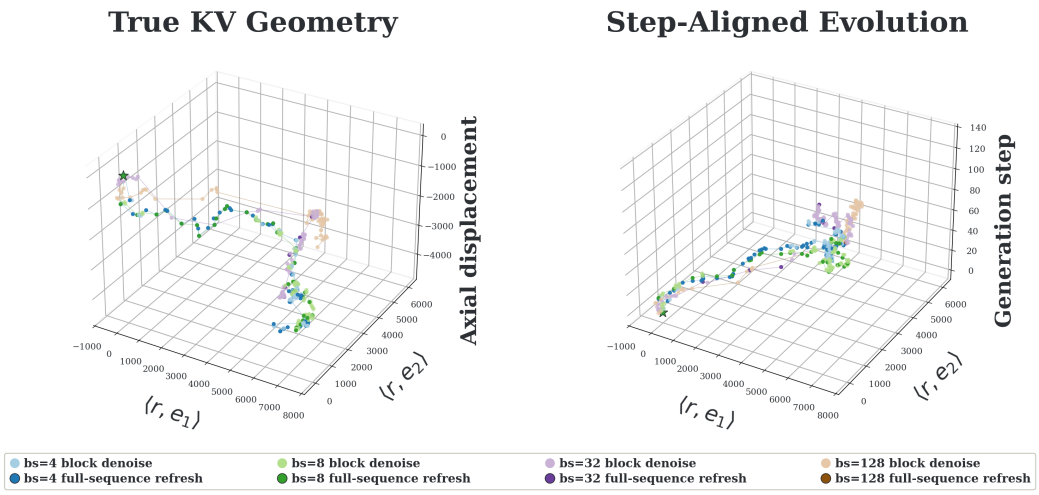


(a) KV-cache trajectory: true KV geometry and step-aligned evolution.

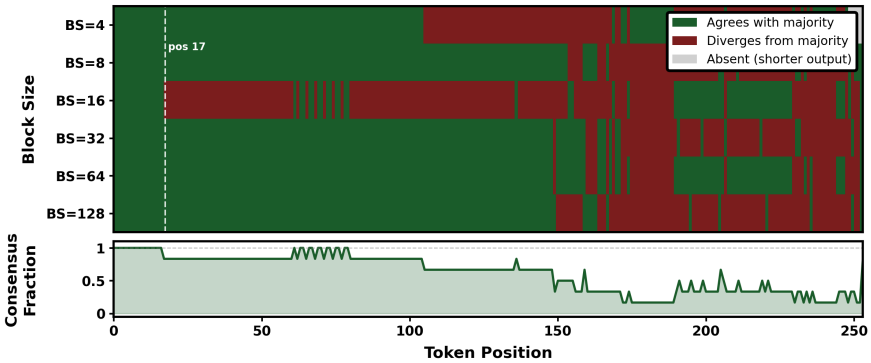


(b) Block-size branch consensus heatmap.

Figure 11: Case study: `HumanEval` sample 15. The same local-region-with-refresh-jumps pattern appears on a longer prompt. Tangent spread accumulates with generation step, while the consensus panel shows progressive divergence of larger block sizes first, consistent with the Lipschitz block bound of Prop. C.2.



(a) KV-cache trajectory: true KV geometry and step-aligned evolution.



(b) Block-size branch consensus heatmap.

Figure 12: Case study: `HumanEval` sample 19. This harder prompt exhibits earlier bifurcation. The tangent-plane fan-out in the KV trajectory and the early color split in the consensus heatmap occur at the same generation step, again confirming that branch bifurcation is the token-level signature of tangent-space separation (Prop. C.4).

Algorithm 1 BLOCKBATCH: Fused Block Batching with Row-Owned KV Cache

Require: Model f_θ, prompt $\mathbf{p}$, generation length G, block sizes $\mathcal{B} = \{b_1, \ldots, b_N\}$, mask token [M], decode threshold τ_{dec}, merge threshold τ_{conf}, sync threshold τ_{sync}, refresh interval R
Ensure: Generated sequence $\hat{\mathbf{x}}$

$L_p \leftarrow |\mathbf{p}|, \quad L \leftarrow L_p + G$
Initialize $\mathbf{X} \in \mathbb{N}^{N \times L}$ with [M]
Set $\mathbf{X}_{k,1:L_p} \leftarrow \mathbf{p}$ for all k
Initialize branch states $S_k \leftarrow (k, b_k, \text{start} = L_p, \text{end} = L_p + b_k, \text{done} = \text{false})$
$(\mathbf{K}, \mathbf{V}), \{\boldsymbol{\ell}^{(k)}\}_{k=1}^{N} \leftarrow$ INITIALFULLFORWARD$(f_\theta, \mathbf{X}, \{S_k\})$ {one full forward; KV broadcast to all rows}
for each active branch S_k **do**
  DECODECURRENTBLOCK$(\mathbf{X}_k, \boldsymbol{\ell}^{(k)}, S_k, \tau_{\text{dec}})$
end for
ADVANCECOMPLETEDBLOCKS$(\mathbf{X}, \{S_k\})$
MERGESYNC$(\mathbf{X}, \mathbf{K}, \mathbf{V}, \{S_k\}, \tau_{\text{conf}}, \tau_{\text{sync}})$
$(\mathbf{K}, \mathbf{V}) \leftarrow$ REFRESHIFNEEDED$(f_\theta, \mathbf{X}, \mathbf{K}, \mathbf{V}, \{S_k\}, R)$
while there exists S_k such that $S_k.\text{done} = \text{false}$ **do**
  $\mathcal{A} \leftarrow$ GETACTIVEBRANCHES$(\mathbf{X}, \{S_k\})$ {unfinished branches with masked tokens in the current block}
  $\mathbf{Q} \leftarrow$ PACKACTIVEBLOCKS$(\mathbf{X}, \mathcal{A})$ {varlen packing removes padded query tokens}
  $\{\boldsymbol{\ell}^{(k)}\}_{S_k \in \mathcal{A}} \leftarrow$ BATCHEDBLOCKFORWARD$(f_\theta, \mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathcal{A})$
  for each $S_k \in \mathcal{A}$ **do**
    DECODECURRENTBLOCK$(\mathbf{X}_k, \boldsymbol{\ell}^{(k)}, S_k, \tau_{\text{dec}})$
    UPDATEPROBABILITYMAP$(S_k, \boldsymbol{\ell}^{(k)})$
    CHECKEOS$(S_k, \mathbf{X}_k, \boldsymbol{\ell}^{(k)})$
  end for
  if there exists an EOS-ready branch **then**
    return SELECTEOSWINNER$(\{S_k\}, \mathbf{X})$
  end if
  ADVANCECOMPLETEDBLOCKS$(\mathbf{X}, \{S_k\})$
  MERGESYNC$(\mathbf{X}, \mathbf{K}, \mathbf{V}, \{S_k\}, \tau_{\text{conf}}, \tau_{\text{sync}})$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ REFRESHIFNEEDED$(f_\theta, \mathbf{X}, \mathbf{K}, \mathbf{V}, \{S_k\}, R)$
end while
$k^\star \leftarrow \arg\max_k S_k.\text{progress}$
return $\mathbf{X}_{k^\star}$

Algorithm 2 MERGESYNC: Cross-branch merge and synchronization

Input: Branch states $\mathcal{S} = \{S_1, \ldots, S_N\}$, token matrix $\mathbf{X}$, unified cache $(\mathbf{K}, \mathbf{V})$
Parameters: merge threshold τ_{conf}, sync threshold τ_{sync}

$S_{\text{lead}} \leftarrow \arg\max_{S_k \in \mathcal{S}} S_k.\text{tokens_decoded}$
if $S_{\text{lead}}.\text{tokens_decoded} = 0$ **then**
  return
end if
$\mathcal{O} \leftarrow$ SORTACTIVEBRANCHES$(\mathcal{S})$
Merge step: fill masked positions using compatible branches.
for each destination branch $S_d \in \mathcal{O}$ **do**
  $\mathcal{C}_d \leftarrow$ COMPATIBLESOURCES$(S_d, \mathcal{O})$
  for each position $i \in$ WINDOWUNION$(\mathcal{C}_d)$ **do**
    if $\mathbf{X}_{d,i} \neq$ [M] **then**
      continue
    end if
    $\mathcal{C}_{d,i} \leftarrow \{S_s \in \mathcal{C}_d : \mathbf{X}_{s,i} \neq \text{[M]}\}$
    if $\mathcal{C}_{d,i} \neq \emptyset$ **then**
      $S_{s^\star} \leftarrow \arg\max_{S_s \in \mathcal{C}_{d,i}} P_d(i, \mathbf{X}_{s,i})$
      $v^\star \leftarrow \mathbf{X}_{s^\star,i}, \quad p^\star \leftarrow P_d(i, v^\star)$
      if $p^\star > \tau_{\text{conf}}$ **then**
        $\mathbf{X}_{d,i} \leftarrow v^\star$
        $S_d.\text{tokens_merged} \leftarrow S_d.\text{tokens_merged} + 1$
      end if
    end if
  end for
  if CURRENTBLOCKDECODED$(S_d, \mathbf{X})$ **then**
    ADVANCEBLOCK(S_d)
  end if
end for
Hard synchronization: copy the leader into lagging branches.
for each active branch $S_d \in \mathcal{S}$ with $S_d \neq S_{\text{lead}}$ **do**
  if $S_{\text{lead}}.\text{tokens_decoded} - S_d.\text{tokens_decoded} > \tau_{\text{sync}}$ **then**
    $\mathbf{X}_{d,:} \leftarrow \mathbf{X}_{\text{lead},:}$
    $(\mathbf{K}_d, \mathbf{V}_d) \leftarrow (\mathbf{K}_{\text{lead}}, \mathbf{V}_{\text{lead}})$
    REALIGNBLOCKWINDOW$(S_d, \mathbf{X}_{d,:})$
    $S_d.\text{tokens_decoded} \leftarrow S_{\text{lead}}.\text{tokens_decoded}$
  end if
end for

Summary. The three case studies provide consistent empirical support for the step / axial decomposition used throughout Sec. C. The axial direction u_0 is moved almost exclusively by full-sequence refreshes (Prop. C.2), the bounded recurrence of Prop. C.3 is visible as tight axial banding between refresh cycles, and the tangent-plane separation predicted by Prop. C.4 aligns step-for-step with the position of first token-level divergence reported by the consensus heatmaps. Together, these observations justify treating block-size branches as competing local traversals of a shared KV-cache geometry rather than independent generations.

D Appendix: Block Batching Algorithm

D.1 Fused Block Batching with a Unified KV Cache

Let [M] denote the mask token, L_p the prompt length, and G the generation length. For each block size $b_k \in \mathcal{B}$, BlockBatch maintains one branch state S_k and one row in the unified token tensor and KV cache.Each branch owns its own token row and KV-cache row; rows are copied across branches only through the merge-and-sync policy.

D.2 Merge and Synchronization Policy

E Appendix: Reproducibility, Artifacts, and Compute Details

E.1 Scientific Artifacts

We use publicly available diffusion language models, benchmark datasets, and evaluation protocols.The model artifacts include LLaDA-Instruct-8B (Nie et al., 2025b) and Dream-Base-7B (Ye et al., 2025b). The benchmark artifacts include GSM8K (Cobbe et al., 2021b), MATH (Hendrycks et al., 2021b), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021b). We cite the original creators of all models and datasets in the main paper and use them only for research evaluation.

E.2 Artifact Licenses and Intended Use

We use existing models and datasets according to their stated research and evaluation purposes.No new human-subject dataset is collected in this work. The experiments are limited to inference-time evaluation and do not involve training on private, personal, or newly collected user data.

E.3 Evaluation Setup

We evaluate GSM8K, MATH, HumanEval, and MBPP using the standard task prompts and metrics. GSM8K uses 5-shot prompting, MATH uses 4-shot prompting, MBPP uses 3-shot prompting, and HumanEval uses 0-shot prompting. HumanEval and MBPP are evaluated with execution-based correctness after code sanitization.

Unless otherwise stated, all methods use greedy decoding with $G = 256$, batch size $= 1$, $\tau_{\text{conf}} = 0.9$. We report accuracy, average NFE, and average latency. Latency is measured as per-example generation wall-clock time and excludes model loading and metric post-processing. For Vanilla decoding, NFE is 256. For Fast-dLLM and LocalLeap, NFE is the average number of model forwards used by the decoding procedure. For BlockBatch, one batched forward pass counts as one NFE; the reported NFE includes initial block denoising, block denoising, and full refresh forwards.

Table 1 compares Vanilla decoding, Fast-dLLM dual cache, LocalLeap, and BlockBatch on LLaDA-1.5-8B, LLaDA-Instruct-8B, and Dream-Base-7B. Fast-dLLM dual cache is evaluated with block size 32. BlockBatch uses $B = \{4, 8, 16, 32, 64, 128\}$, $\tau_{\text{sync}} = 8$, $R = 32$, and $\tau_{\text{merge}} = 0.5$. LocalLeap uses its default script configuration: `threshold` $= 0.9$, `radius` $= 4$, with `relaxed_threshold` $= 0.75$ for LLaDA models and `relaxed_threshold` $= 0.8$ for Dream-Base-7B.

For Table 2, we ablate the BlockBatch synchronization threshold over $\tau_{\text{sync}} \in \{4, 8, 16, 32, 64\}$. All other decoding settings match the Table 1 BlockBatch configuration. The table reports GSM8K and HumanEval results for LLaDA-Instruct-8B and Dream-Base-7B.

For Table 3, we ablate the full-refresh interval over $R \in \{4, 8, 16, 32, 64, 128\}$, while fixing $\tau_{\text{sync}} = 8$, $\mathcal{B} = \{4, 8, 16, 32, 64, 128\}$.

E.4 Computational Infrastructure and Budget

Latency and throughput profiling experiments were run on NVIDIA H200 GPUs.