EARLY-BIRD DECODING: ACCELERATING DIFFUSION LLMS WITH LEARNABLE BLOCK SIZES AND PARALLEL SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion large language models (dLLMs) offer a promising parallel decoding paradigm as an alternative to autoregressive generation through iterative unmasking. However, dLLMs typically require many steps before token confidence reaches the decoding threshold, resulting in inefficient inference even with block-wise KV caching. To accelerate dLLM inference, we *for the first time* propose an "early-bird (EB)" decoding framework, motivated by the observation that tokens with similarly low entropy tend to cluster and can be jointly decoded earlier, before reaching the confidence threshold. In particular, our EB-Decode framework integrates two key enablers: (1) a learnable network that adaptively groups tokens with similar uncertainty into variable-length blocks, rather than relying on fixed block sizes; (2) a position-aware sampler that learns to unmask tokens in parallel using fewer decoding steps within predicted variable-length blocks. Both components are developed without modifying pretrained dLLM weights and can therefore be directly deployed as plug-ins during serving, with negligible training and inference overhead. Extensive experiments across three models and four benchmarks consistently validate our observation and the effectiveness of EB-Decode, achieving **3.53–18.76**$\times$ higher throughput than vanilla decoding with comparable accuracy and up to **1.58**$\times$ higher throughput than the baseline with the highest average speedup, Fast-dLLM. We will release the code upon acceptance.

1 INTRODUCTION

Diffusion large language models (dLLMs) have emerged as a compelling alternative to autoregressive (AR) models, breaking the sequential bottleneck of token-by-token decoding by generating text in parallel through iterative denoising over masked positions (Austin et al., 2021a; Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025). This paradigm enables bidirectional context utilization and makes dLLMs a practical and increasingly scalable approach for fast, high-quality text generation, as demonstrated by large-scale systems such as Gemini Diffusion (Google DeepMind, 2025), Seed-Diffusion (Song et al., 2025), and Mercury (Khanna et al., 2025). However, current open-source dLLMs such as LLaDA (Nie et al., 2025; Zhu et al., 2025b; Bie et al., 2025) and Dream (Ye et al., 2025) still struggle to consistently deliver better accuracy-efficiency tradeoffs compared to their AR counterparts of similar size. For example, the AR model LLaMA3-8B-Instruct achieves 48.0 tokens/s, whereas the dLLM LLaDA-8B-Instruct reaches only 3.5 tokens/s under an NVIDIA A100-PCIe 40GB GPU setup. Its accuracy also drops from 83.5% at 1,024 denoising steps to 54.1% at 256 denoising steps (Wang et al., 2025c; Peng et al., 2025; Kang et al., 2025; Li et al., 2026), which limits the widespread adoption of dLLMs in real-world deployments.

Recently, several works have attempted to mitigate the inference efficiency challenges of dLLMs, including semi-AR variants for block-wise sequential generation (Arriola et al., 2025; Cheng et al., 2025), Key-Value (KV) and activation caching (Wu et al., 2025; Liu et al., 2025; Ma et al., 2025a; Hu et al., 2025; Chen et al., 2025a), efficient sampling algorithms (Huang et al., 2025; Bao et al., 2025; Wei et al., 2025; Lu et al., 2025; Israel et al., 2025; Wang et al., 2025b; Zhang et al., 2026b; Luo et al., 2026; Luxembourg et al., 2025), and step distillation (Hayakawa et al., 2024; Deschenaux & Gulcehre, 2024; Chen et al., 2025b; Liang et al., 2026; Zhang et al., 2026a). However, most of them rely on fixed block selection and confidence thresholds during decoding, implicitly assuming uniform token difficulty along the sequence, which wastes parallelism on hard tokens and under-utilizes it

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.

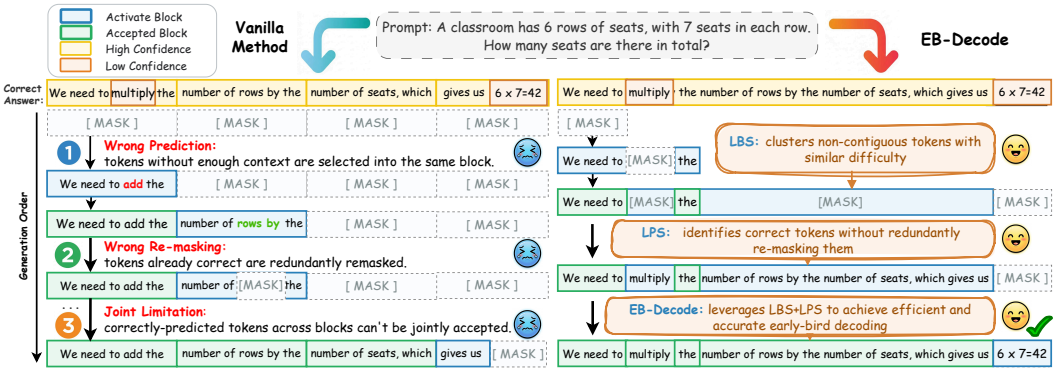


Figure 1: Conceptual comparison illustrating the differences between vanilla block-wise decoding (Left) (Arriola et al., 2025) and the proposed EB-Decode framework (Right).

on easy ones, contradicting the heterogeneous nature of natural language. For example, in code, function signatures are typically easier to predict than function bodies, and in mathematical reasoning, routine arithmetic is straightforward whereas multi-step derivations are more challenging. Beyond that, a few recent methods introduce simple heuristics for adaptive block sizing and thresholding, such as delimiter (e.g., period) detection (Lu et al., 2025), sliding windows (Luo et al., 2026), or entropy-based boundaries (Zhang et al., 2026b). Some methods attempt to learn parallel decoding strategies (Bao et al., 2025), but still rely on fixed block sizes. These limitations call for a principled framework that not only adaptively determines block sizes but also inherently enables early decoding under homogeneous token difficulty within each block.

In this work, we *for the first time* propose a principled "early-bird" decoding framework, motivated by our observation that tokens with similar low entropy and close semantic meanings tend to cluster and can be jointly decoded much earlier before reaching the confidence threshold, due to their similar difficulty levels. To enable such principled EB decoding, two key challenges arise: **First**, how can we automatically learn adaptive block sizes instead of relying on heuristics? Unlike prior delimiter-based methods, our observation shows that token entropy exhibits a staircase pattern across token positions, indicating that at certain key steps, adjacent or even non-contiguous tokens share similar entropy or uncertainty. This motivates the design of a lightweight network that leverages entropy and positional semantics to dynamically predict block sizes on the fly. **Second**, how can we decode the predicted block of tokens earlier before reaching the confidence threshold? Our observation shows that confidence-based decoding wastes computation on tokens that have already converged: many tokens become correct in early denoising steps but are repeatedly remasked because their confidence has not yet reached the threshold. Moreover, under fixed block sizes, confident tokens outside the current block are excluded from early finalization. These observations motivate a position-aware, learnable parallel sampler that leverages per-token statistics (i.e., entropy, position, and step) to determine which positions can be finalized at earlier steps, while naturally supporting variable-length blocks. To the best of our knowledge, this work is the first to tackle the above challenges toward a principled EB decoding framework. Our contributions are summarized as follows:

- We propose a principled early-bird decoding framework for efficient dLLM inference acceleration, termed **EB-Decode**, that automatically clusters tokens with similar difficulty into blocks and enables their early decoding with fewer denoising steps than confidence-based methods.
- *Enabler 1*: We adopt a **learnable block size (LBS)** prediction network to dynamically cluster contiguous or non-contiguous tokens with similar difficulty and semantic coherence on the fly.
- *Enabler 2*: We introduce a position-aware **learnable parallel sampling (LPS)** network to identify and finalize converged tokens at early denoising steps under variable-length blocks.
- Extensive experiments across three dLLMs and four representative benchmarks demonstrate the effectiveness of EB-Decode: it achieves **3.53–18.76**× throughput improvement over the vanilla decoder with comparable accuracy and only ∼**6.21%** routing overhead, and delivers an additional **1.20**× speedup when combined with KV-cache pipelines.

2 Related Works

Efficient Inference of dLLMs. Appendix A reviews dLLMs and their representative models. Recent work improves dLLM decoding along several directions. Cache-based methods reuse key-value

projections across denoising steps to avoid redundant computation (Liu et al., 2025; Wu et al., 2025; Ma et al., 2025a; Hu et al., 2025). Distillation-based methods train the model to commit more tokens per step (Chen et al., 2025b; Liang et al., 2026). Efficient sampling methods redesign the denoising trajectory to better allocate computation (Wei et al., 2025; Israel et al., 2025; Bao et al., 2025; Ma et al., 2025b; Li et al., 2025). Dynamic block methods resize the block at runtime by aligning boundaries with confidence or entropy shifts (Lu et al., 2025; Zhang et al., 2026b; Luo et al., 2026). An orthogonal line targets test-time quality through revocable draft-and-verify decoding (Hong et al., 2025), temporal-dynamics voting (Wang et al., 2025b), and inference-time remasking (Wang et al., 2025a). In contrast, our proposed EB-Decode explicitly leverages the token-level statistics, and learns to dynamically cluster tokens with similar difficulty and adapt the sampling schedule to identify and finalize converged tokens at early denoising steps under variable-length blocks.

Early-Bird (EB) Phenomenon. Early prediction is important for efficient training and inference. For training, the EB ticket hypothesis (You et al., 2019) shows that small subnetworks (i.e., lottery tickets (Frankle & Carbin, 2018)) can be identified early in training via mask-distance convergence, achieving accuracy comparable to overparameterized networks. This EB phenomenon has been consistently observed in BERT (Chen et al., 2021b), GCNs (You et al., 2022), LLMs (Gu et al., 2024), and diffusion models (Whalen et al., 2025). For inference, early-exit mechanisms enable dynamic computation by terminating inference once predictions become sufficiently confident. They were first applied to RNNs through a learned halting unit (Graves, 2016), and later generalized to CNNs (Teerapittayanon et al., 2016) and Transformers (Xin et al., 2020; Zhou et al., 2020; Schuster et al., 2022), where intermediate layers can halt computation once a confidence or prediction-agreement criterion is satisfied. In this work, we *for the first time* observe the EB phenomenon during the decoding phase in dLLMs and leverage it to enable efficient EB decoding.

3 The Observations of EB Phenomenon in dLLM Decoding

dLLMs generate text by iteratively unmasking tokens, typically decoding fixed-size blocks left to right and committing tokens whose confidence exceeds a threshold (formulation in Appendix B). To understand the inefficiency of existing dLLM decoding, we analyze the denoising trajectory and identify two empirical phenomena overlooked by fixed-block, threshold-based decoding:

Observation 1: Staircase Entropy Pattern. To examine how token difficulty varies across positions, we measure the Shannon entropy H_t^i at each masked position i and denoising step t. We visualize this on a GSM8K sample with generation length 256, decoded by the block-wise diffusion method (Wu et al., 2025).

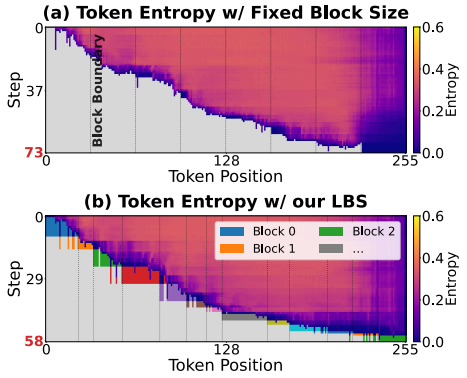
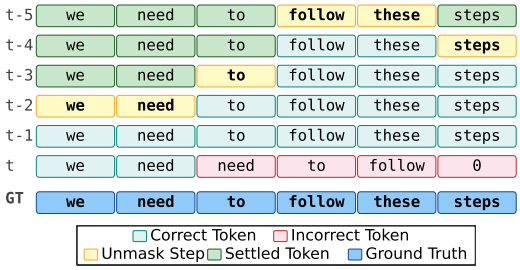


Figure 2: Comparison of entropy heatmaps between **(a)** fixed block size and **(b)** our non-contiguous learnable block size method, where each colored region represents a block. Gray regions mark unmasked tokens.

As shown in Fig. 2 (a), the entropy does not decrease smoothly across token positions. Instead, it exhibits a staircase pattern, consisting of long, flat low-entropy plateaus that typically correspond to predictable syntax or boilerplate content, such as reasoning connectives like *"To solve this problem, we first need to ..."* in mathematical reasoning tasks, or structural scaffolding around known function signatures like `def function_name(args):` in code generation tasks. This suggests that semantically coherent tokens within each plateau share similar difficulty, and these regions exhibit similarly low entropy across denoising steps. However, this conflicts with fixed block boundaries, under which such semantically coherent regions are split in a manner agnostic to semantic structure, preventing stabilized regions across block boundaries from being decoded jointly and leaving the available parallelism unexploited. In addition, these flat plateaus are occasionally interrupted by short high-entropy spikes corresponding to harder-to-predict tokens. Although such hard tokens sit within an otherwise semantically coherent region, they are not well-suited to being decoded jointly with the surrounding easy tokens. Yet fixed-block methods enforce a strict left-to-right order that forces these hard tokens to be resolved inside the current block before any subsequent content can be generated. In contrast, as shown in Fig. 2 (b), a block selection strategy that allows non-contiguous grouping could defer these hard tokens to a later block, letting the model first decode the surrounding easy context and leverage the additional right-side information when eventually resolving them.

(a) Per-token Denoising Trajectory



(b) Block-relative Lead-time on GSM8K

	Vanilla	Confidence-based	**EB-Decode (Ours)**
Mean Time	13.4 Steps	2.3 Steps	**0.7 Steps**
Fraction of Tokens			
$\geq$1 Step	96.3%	62.5%	**25.1%**
$\geq$3 Steps	88.3%	35.0%	**10.5%**
$\geq$5 Steps	80.2%	18.1%	**4.6%**
$\geq$10 Steps	60.5%	3.5%	**0.7%**

Figure 3: **(a)** Per-token denoising trajectory, where each column represents a token position across steps within one segment. Light green cells indicate tokens whose predictions become correct before they are committed, illustrating the inefficiency of confidence-based decoding. **(b)** Mean lead-time (in denoising steps) and fraction (%) of tokens with lead $\geq k$ denoising steps before commit, for vanilla decoding (Nie et al., 2025), confidence-based decoding (Wu et al., 2025), and EB-Decode.

Observation 2: Early Convergence Phenomenon. Even when block selections are well aligned with token difficulty, decoding still has to decide *when* each token within a block should be committed. To examine this, we track the per-step prediction at each masked position for six representative token positions of a GSM8K sample decoded using the vanilla block-wise method (Nie et al., 2025). As shown in Fig. 3 (a), we observe that many tokens reach the correct prediction several denoising steps before their per-token confidence crosses the threshold, in principle allowing them to be committed early. These early-converged tokens, however, are repeatedly remasked before being finalized, introducing step-level redundancy. Fig. 3 (b) quantifies this redundancy by showing that under both the vanilla (Nie et al., 2025) and confidence-based method (Wu et al., 2025), many tokens remain unaccepted for multiple denoising steps after their predictions become correct. As reported in Fig. 3 (b), the mean lead times of the two baseline methods are 13.4 and 2.3 steps, respectively. The fractions of tokens are 96.3% and 62.5% when the lead time is ≥ 1 step, and 60.5% and 3.5% when the lead time is ≥ 10 steps. This behavior arises because existing methods evaluate each masked position independently and cannot distinguish correct-but-uncertain predictions from genuinely incorrect ones, highlighting the need for a principled and learnable strategy to commit correct tokens earlier.

4 The Proposed EB-Decode Framework

Overview. The two observations in Sec. 3 reveal two fundamental issues in current dLLM decoding: (1) a fixed block size that ignores the staircase pattern of token difficulty and the non-contiguous nature of tokens that should be grouped, and (2) a fixed confidence threshold that overlooks the adaptive nature of how early predictions actually converge. As illustrated in Fig. 4 (a), our proposed EB-Decode framework replaces two static choices with learnable counterparts while keeping the base dLLM frozen. Specifically, LBS replaces the fixed block size and decides *where* to decode at each step, as described in Sec. 4.1, while LPS replaces the fixed confidence threshold and decides *which* positions within a given variable-length block to finalize early, as described in Sec. 4.2. Both modules reuse the base dLLM's forward pass and introduce negligible training and inference overhead.

4.1 Enabler 1: Learnable Block Size (LBS)

The staircase entropy pattern identified in Sec. 3 reveals that token difficulty is spatially clustered: low-entropy regions alternate with sharp uncertainty spikes that do not align with fixed block boundaries. A fixed partition either splits an easy plateau across two blocks, wasting a decoding pass, or lumps easy and hard tokens together, causing hard tokens to bottleneck their easier neighbors. To resolve this, we introduce LBS, a lightweight router $\mathcal{R}_\phi$ that at each step selects a variable-length, non-contiguous active block $\mathcal{B} \subseteq \mathcal{M}$ from the currently masked positions $\mathcal{M} = \{i : x^i = \texttt{[MASK]}\}$.

LBS Router. As shown in Fig. 4 (b), for each masked position i, the router takes two complementary inputs: (1) *Normalized entropy*. Let p^i denote the predictive distribution of the frozen dLLM over the vocabulary $\mathcal{V}$ at masked position i, with $v \in \mathcal{V}$ a vocabulary token. The normalized entropy $\tilde{H}^i = (-\sum_{v\in\mathcal{V}} p^i(v) \log p^i(v))/\log|\mathcal{V}|$ measures local prediction uncertainty. Entropy alone, however, does not distinguish between qualitatively different sources of uncertainty: a function word may be uncertain among a few interchangeable alternatives, while a rare content word may be uncertain because its identity depends on context not yet resolved—two situations that call for different routing decisions; (2) *Top-1 predicted token ID* $\hat{x}^i = \arg\max_{v\in\mathcal{V}} p^i(v)$, i.e., the most likely

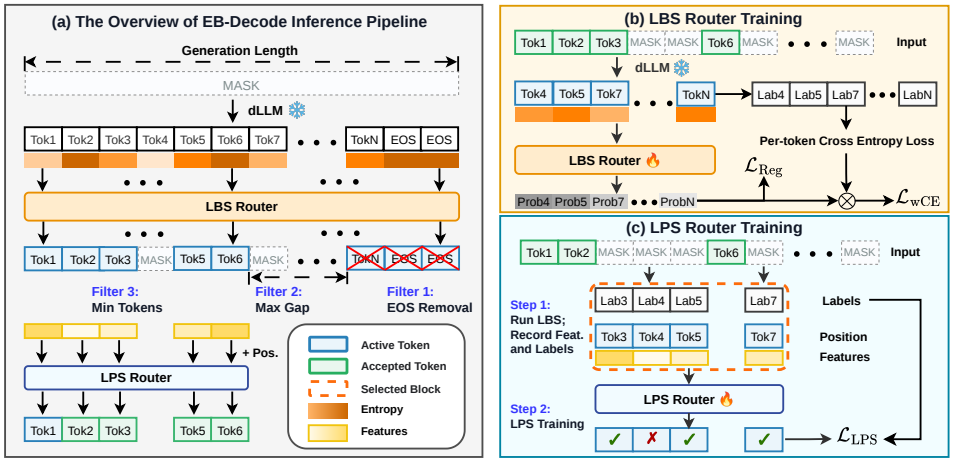


Figure 4: Overview of the proposed EB-Decode framework.

token at position i. Inspired by (Lu et al., 2025), we embed token IDs using the frozen base-model token embedding table. This allows the router to differentiate between semantically trivial tokens and semantically informative ones. In summary, the router maps these input signals to a per-masked-token inclusion probability p_ϕ^i of whether a masked token should be included in the final selected block:

$$p_\phi^i = \sigma(O_\phi^i), \quad O_\phi^i = \mathcal{R}_\phi(\tilde{H}^i, \hat{x}^i), \quad i \in \mathcal{M}, \tag{1}$$

where O_ϕ^i is the router output logit and σ is the sigmoid function. The router is implemented as a lightweight two-layer Transformer encoder, with full architectural details described in Appendix C.1.

LBS Router Training. We train the LBS router in a block-wise masking style. A single frozen forward pass through the dLLM yields entropy $\tilde{H}^i$ and token ID $\hat{x}^i$ at every masked position, which are then passed into the LBS router to get the per-token probability p_ϕ^i. We train the router by minimizing a weighted cross-entropy loss ($\mathcal{L}_{\text{wCE}}$) with a regularization term ($\mathcal{L}_{\text{Reg}}$):

$$\mathcal{L}_{\text{LBS}} = \mathcal{L}_{\text{wCE}} - \mathcal{L}_{\text{Reg}} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \sigma(O_\phi^i) \cdot \ell_{\text{CE}}^i \; - \; \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \sigma(O_\phi^i) \tag{2}$$

where ℓ_{CE}^i denotes the per-masked-token cross-entropy loss of the frozen dLLM at position i. The first term $\mathcal{L}_{\text{wCE}}$ trains the router to assign high inclusion probability (i.e., likelihood of being included in the current block) to positions where the dLLM already predicts correctly (i.e., low ℓ_{CE}^i), and assign low inclusion probability to positions where the dLLM is either incorrect or underconfident (i.e., high ℓ_{CE}^i). The second term acts as a regularization role that prevents the degenerate solution of selecting no tokens—which would trivially minimize the first term—by rewarding larger total inclusion. Because the regularizer is subtracted, the two terms pull in opposite directions, which stabilizes the block size. The training procedure is also summarized in Alg. 1; the full pseudocode is detailed in Appendix C.2.

LBS Router Inference. As shown in Alg. 2, at each step, entropy $\tilde{H}^i$ and token ID $\hat{x}^i$ are computed from the dLLM output logits over masked positions and passed to $\mathcal{R}_\phi$, producing a candidate block $\mathcal{C} = \{i : p_\phi^i > 0.5\}$. This candidate block is then refined by three filters, each addressing a specific failure mode. (1) *EOS removal*. End-of-sequence tokens have near-zero entropy, so the router nearly always selects them. However, unmasking EOS tokens too early in the inference process would truncate generation prematurely and severely damages accuracy on tasks that require step-by-step reasoning like math and coding (Huang et al., 2025). Thus, we remove all EOS tokens from $\mathcal{C}$ entirely. We quantify the dominance of EOS predictions at the sequence tail in Appendix I. (2) *Max-gap constraint* ($g_{\max}$). Ideally, we want to identify a homogeneous difficulty plateau as the block, instead of cherry-picking isolated easy tokens scattered across a hard region. If consecutive selected positions are separated by more than $g_{\max}$ tokens, the candidate block straddles an unresolved hard region, violating the spatial coherence assumption and forcing the model to denoise positions that have not yet been committed. Thus, starting from the leftmost selected position, any candidate whose gap to its predecessor exceeds $g_{\max}$ is discarded along with all subsequent selected positions. (3) *Min-tokens constraint* ($L_{\min}$). After applying the above two filters, if the resulting candidate set contains fewer

Algorithm 1 EB-Decode Training

Require: Frozen dLLM $\mathcal{F}_\theta$, prompt-response pair $\mathcal{D}$

// LBS router $\mathcal{R}_\phi$

for $x \in \mathcal{D}$ **do**
  $\tilde{H}^i, \hat{x}^i \leftarrow \mathcal{F}_\theta(\text{Mask}(x))$ ▷ LBS Inputs
  $O^i_\phi \leftarrow \mathcal{R}_\phi(\tilde{H}^i, \hat{x}^i)$
  Update ϕ via $\mathcal{L}_{\text{LBS}}$ ▷ Eq. (2)
end for

// LPS router $\mathcal{R}_\psi$

for $x \in \mathcal{D}$ **do**
  Decode with LBS; record trace $\mathcal{T}$
end for
for $(f^i, \text{pos}^i) \subset \mathcal{T}$ **do** ▷ LPS Inputs
  $O^i_\psi \leftarrow \mathcal{R}_\psi(f^i, \text{pos}^i)$
  Update ψ via $\mathcal{L}_{\text{LPS}}$ ▷ Eq. (3)
end for
return $\mathcal{R}_\phi, \mathcal{R}_\psi$

Algorithm 2 EB-Decode Inference

Require: $\mathcal{F}_\theta$, $\mathcal{R}_\phi$, $\mathcal{R}_\psi$, and thresholds τ, τ_ψ

$x \leftarrow$ Prompt & `[MASK]`
while any $x^i =$ `[MASK]` **do**
  $\tilde{H}^i, \hat{x}^i, f^i \leftarrow \mathcal{F}_\theta(x)$
  $f^i = (\kappa^i, \tilde{H}^i, \Delta^i, \rho)$ ▷ Feature Details
  // LBS
  if no active block **then**
    $\mathcal{B} \leftarrow \text{Filters}\big(\mathcal{R}_\phi(\tilde{H}^i, \hat{x}^i)\big)$
  end if
  // LPS
  $O^i_\psi \leftarrow \mathcal{R}_\psi(f^i, \text{pos}^i)$
  $\mathcal{A} \leftarrow \{i \in \mathcal{B} : \kappa^i \geq \tau \vee p^i_\psi > \tau_\psi\}$
  $x^i \leftarrow \hat{x}^i$ for $i \in \mathcal{A}$ ▷ Accept & Commit
  If $\mathcal{B}$ resolved, move to Line 5
end while
return x

than $L_{\min}$ tokens, LBS falls back to selecting the first 32 masked positions in left-to-right order. This constraint is critical for accuracy. If the number of tokens is too small after filtering, the router can degenerate to selecting scattered easy tokens anywhere in the sequence. In summary, the router makes per-token decisions, resulting in an inherently non-contiguous candidate block that adapts to the entropy pattern. The filters then impose contiguity at inference time.

4.2 Enabler 2: Learnable Parallel Sampling (LPS)

LBS adapts block boundaries to local difficulty but keeps the commit rule fixed, leaving intra-block parallelism underutilized. The early convergence observation in Sec. 3 shows that a masked position's top-1 prediction often becomes correct several steps before its confidence reaches the threshold, making additional refinement steps unnecessary. To bridge this gap, we introduce LPS, a lightweight router $\mathcal{R}_\psi$ that learns when a masked position within a variable-length block is ready to be finalized, enabling early commitment before reaching the confidence threshold, as shown in Fig. 4 (c).

Position-aware LPS Router. Unlike many existing decoding methods (Nie et al., 2025; Ye et al., 2025; Wu et al., 2025; Lu et al., 2025) that rely on top-1 confidence κ^i as a static commit criterion, LPS learns this decision, using confidence as the primary input and augmenting it with additional signals. First, to make the LPS router position-aware, we incorporate the token's normalized intra-block position embedding, $\text{pos}^i = {}^{(i-b)}/_{|\mathcal{B}|} \in [0, 1]$, where b is the left boundary index and $|\mathcal{B}|$ is the block size. This encoding captures the token's relative location within the block and enables the router to correlate decisions across positions rather than scoring them independently. To support variable-length block inputs, our LPS router adopts a two-layer Transformer architecture to leverage positional information, rather than an MLP-based design (Bao et al., 2025) that cannot handle variable-length inputs. Model details are provided in Appendix D.1. Second, to enable the LPS router to distinguish correct-but-underconfident tokens from genuinely uncertain ones, we incorporate three auxiliary features: the normalized entropy $\tilde{H}^i_t$, the top-1/top-2 confidence gap Δ^i_t, and the block-wise mask ratio ρ_t. Appendix D.1 explains the role of each feature. For each masked position i in the active block $\mathcal{B}$, the router takes $f^i_t = (\kappa^i_t, \tilde{H}^i_t, \Delta^i_t, \rho_t)$ and pos^i as input.

Self-supervised LPS Router Training. We train LPS using a *generate-then-replay* pipeline that requires no human label. We first run the base dLLM under its default decoding schedule and record the resulting completed sequence x_0. We treat x_0 as a convergence target rather than absolute ground truth, so the LPS router learns to predict when a token has stabilized to the model's own final prediction. During replay, at each intermediate step t, we recompute the features f^i_t and assign $y^i_t = \mathbf{1}[\hat{x}^i_t = x^i_0]$, indicating whether the current top-1 prediction already matches x_0. To keep the replay aligned with the original generation trajectory, we apply an oracle rule that unmasks positions where the current prediction matches x_0, and replaces mismatched positions with their corresponding tokens from x_0. As prematurely committing an incorrect token is irreversible, we use a weighted binary cross-entropy loss to penalize false positives more heavily than false negatives during training:

$$\mathcal{L}_{\text{LPS}} = \frac{1}{|\mathcal{M}_t|} \sum_{i \in \mathcal{M}_t} w^i_t \cdot \text{BCE}\Big(\sigma\big(\mathcal{R}_\psi(f^i_t, \text{pos}^i_t)\big), y^i_t\Big), \tag{3}$$

where σ is the sigmoid function, and negative samples (incorrect early commitments) receive weights $w_t^i > 1$. The training procedure is briefly summarized in Alg. 1, with full trace-generation and training details provided in Appendix D.2 and D.3.

LPS Router Inference. As shown in Alg. 2, at each decoding step, the base dLLM produces top-1 predictions and confidences for all masked positions. Within the active block $\mathcal{B}$ selected by LBS, a position i is committed if its base confidence exceeds the threshold τ or its LPS score exceeds a threshold τ_ψ tuned on a held-out validation split. In this way, LPS preserves the standard threshold rule for tokens the model is already confident about, while committing correct-but-underconfident tokens that would otherwise be deferred. If no position in $\mathcal{B}$ satisfies either condition, we fall back to committing the highest-confidence masked position to prevent decoding from stalling, following standard practice in confidence-based decoding. Once $\mathcal{B}$ is fully resolved, LBS selects the next block. Full pseudocode is provided in Appendix E.

5 Experiments

5.1 Experimental Setup

Models, Benchmarks, and Metrics. *Models.* We evaluate EB-Decode on three open-source dLLMs: LLaDA-8B-Instruct (Nie et al., 2025), Dream-v0-Instruct-7B (Ye et al., 2025), and LLaDA-1.5 (Zhu et al., 2025a). *Benchmarks.* Our evaluation covers mathematical reasoning on GSM8K (Cobbe et al., 2021) (0-shot) and MATH500 (Lightman et al., 2023) (0-shot), as well as code generation on HumanEval (Chen et al., 2021a) (0-shot) and MBPP (Austin et al., 2021b) (3-shot). For GSM8K and MATH500, we append a chain-of-thought prompt suffix (Wei et al., 2022) to each question following (Chen et al., 2025b). *Metrics.* We report task accuracy, throughput, and the associated speedups compared to the base models. Detailed experimental settings are provided in Appendix F.

Baselines. We compare EB-Decode against four inference-stage baselines: (1) the vanilla block-wise decoding method used by each base dLLM (Nie et al., 2025; Ye et al., 2025; Zhu et al., 2025a), which serves as the no-acceleration reference; (2) Fast-dLLM (Wu et al., 2025), which commits multiple tokens per step based on a confidence threshold (for a fair comparison, we use its parallel decoding without KV cache, and report its results with KV cache in Tab. 6); (3) AdaBlock-dLLM (Lu et al., 2025), which adaptively places block boundaries using delimiter-token confidence; and (4) Learn2PD (Bao et al., 2025), which employs a lightweight filter for parallel decoding. A detailed comparison between EB-Decode and these and related methods is given in Appendix H.

Router Training Details. Both routers are lightweight, with about 0.6M trainable parameters for LBS and 0.03M for LPS. The LBS router is trained on dLLM-generated responses from $92{,}000$ prompts randomly sampled from GSM8K (Cobbe et al., 2021), the PRM12K (Lightman et al., 2023), and the Numina-Math dataset (Li et al., 2024), following (Chen et al., 2025b). The router is trained for 6 epochs using AdamW with a learning rate of $5\mathrm{e}{-5}$. The actual training of the LBS router takes only about 2 hours. The LPS router is trained on 10K prompts randomly sampled from AQUA-RAT (Ling et al., 2017). Training uses AdamW with a learning rate of $1\mathrm{e}{-3}$ and is early-stopped when the validation recall does not improve for 100 epochs. Trace generation takes about 2 hours, while router training itself requires only around 10 minutes. Further details are provided in Appendix C and D.

5.2 Comparison of EB-Decode with SOTA Baselines

Tab. 1 summarizes the comparison between our proposed EB-Decode and four baselines in terms of accuracy and throughput across three dLLMs and four benchmarks, measured on H100 GPUs. We observe that both LBS and LPS in our EB-Decode framework consistently achieve better accuracy-efficiency trade-offs. EB-Decode with LBS alone already improves throughput across all settings, while even improving accuracy in several cases. For example, on LLaDA-8B-Instruct with HumanEval, LBS increases throughput from 23.70 to 68.83 TPS while simultaneously improving accuracy from 43.90% to 45.73%. This result suggests that adaptive block boundaries not only unlock greater parallelism, but also provide difficult tokens with richer right-context information during decoding. The full EB-Decode improves throughput by **3.53–18.76**$\times$ over the vanilla decoder, while maintaining comparable accuracy (around $\pm 1\%$). EB-Decode also outperforms the strongest baseline in every setting, delivering up to 1.58$\times$ higher throughput than Fast-dLLM while maintaining comparable or even higher accuracy. These results demonstrate the effectiveness of our EB-Decode.

5.3 Ablation Studies of EB-Decode

Table 1: Comparison of EB-Decode and baseline methods across four benchmarks and three models. We report tokens per second per GPU (TPS), speedup (Sp.up), and accuracy (Acc.%). Speedup is measured relative to the vanilla decoding method. The best throughput and speedup are highlighted in **bold**. "—" indicates not applicable, as Learn2PD provides no implementation for LLaDA-1.5.

	LLaDA-8B-Instruct			Dream-v0-Instruct-7B			LLaDA-1.5		
Method	**Sp.up**	**TPS**	**Acc.(%)**	**Sp.up**	**TPS**	**Acc.(%)**	**Sp.up**	**TPS**	**Acc.(%)**
HumanEval									
Vanilla	1.00×	23.70	43.90	1.00×	11.33	54.88	1.00×	23.79	43.90
Fast-dLLM	2.66×	63.08	43.90	4.29×	48.57	53.66	2.72×	64.60	42.68
AdaBlock-dLLM	2.38×	56.33	42.68	4.05×	45.88	52.44	2.52×	59.95	41.46
Learn2PD	2.78×	65.92	42.68	2.15×	24.31	53.66	—	—	—
EB-Decode (LBS)	2.90×	68.83	45.73	4.46×	50.55	55.49	2.68×	63.73	41.46
EB-Decode (LBS+LPS)	**3.53**×	**83.69**	43.29	**5.61**×	**63.53**	54.88	**3.58**×	**85.19**	42.68
MBPP									
Vanilla	1.00×	11.95	40.00	1.00×	2.23	53.40	1.00×	8.28	40.80
Fast-dLLM	4.13×	49.31	40.20	11.88×	26.49	53.00	6.41×	53.13	40.20
AdaBlock-dLLM	4.02×	48.02	41.20	11.54×	25.74	52.40	6.01×	49.82	40.60
Learn2PD	4.72×	56.37	40.00	4.04×	9.01	52.00	—	—	—
EB-Decode (LBS)	4.47×	53.37	40.40	14.70×	32.77	54.80	6.39×	52.94	40.60
EB-Decode (LBS+LPS)	**5.37**×	**64.18**	40.20	**18.76**×	**41.83**	53.60	**7.76**×	**64.26**	39.80
GSM8K-CoT									
Vanilla	1.00×	17.04	79.07	1.00×	12.03	77.78	1.00×	16.94	79.76
Fast-dLLM	4.65×	79.20	78.70	4.51×	54.20	77.86	5.15×	87.27	79.45
AdaBlock-dLLM	4.18×	71.22	79.00	4.57×	54.97	78.70	4.89×	82.87	79.00
Learn2PD	4.91×	83.64	78.24	2.07×	24.96	78.09	—	—	—
EB-Decode (LBS)	5.09×	86.80	79.68	4.92×	59.22	78.10	4.79×	81.20	80.36
EB-Decode (LBS+LPS)	**6.14**×	**104.55**	78.24	**6.36**×	**76.47**	77.33	**6.25**×	**105.91**	79.53
MATH500-CoT									
Vanilla	1.00×	21.72	42.40	1.00×	24.97	51.00	1.00×	15.10	43.20
Fast-dLLM	3.26×	70.77	42.00	3.10×	77.39	51.40	5.00×	75.59	42.80
AdaBlock-dLLM	3.07×	66.63	41.60	3.08×	77.00	48.80	4.69×	70.88	42.20
Learn2PD	3.78×	82.11	41.80	1.12×	28.05	49.80	—	—	—
EB-Decode (LBS)	3.32×	72.03	41.20	2.96×	73.91	49.80	4.92×	74.24	42.60
EB-Decode (LBS+LPS)	**3.86**×	**83.79**	42.40	**3.57**×	**89.06**	49.40	**5.55**×	**83.79**	42.40

LBS Design Choices. To isolate the contribution of the learned block selection, Tab. 2 removes the LBS router and selects blocks randomly, while keeping the three filters of Sec. 4.1. The filters raise accuracy from 49.13% to 76.80%, but this variant still falls 2.88 points below LBS and runs $3.62\times$ slower (23.96 vs. 86.80 TPS), so the filters alone cannot replace the learned router. Conversely, Tab. 8 in Appendix C.3 shows that each filter also contributes when added on top of the LBS router. We also train an LBS router of comparable size on LLaDA-8B-Instruct that takes only normalized entropy as input. Its converged training loss is higher, and accuracy across the four benchmarks drops by 1.2 points on average at similar TPS, suggesting that the semantic information from predicted token IDs helps the LBS router select better blocks.

Table 2: Ablation of the LBS router on LLaDA-8B-Instruct and GSM8K-CoT.

Variant	**Acc.(%)**	**TPS**
w/o LBS router	49.13	11.56
+ EOS removal	68.76	24.24
+ Max-gap constraint	76.27	23.16
+ Min-tokens constraint	76.80	23.96
LBS router + all filters (ours)	**79.68**	**86.80**

LPS Design Choices. Tab. 3 additively builds up the LPS router on top of LBS selected blocks. Starting from an MLP taking only confidence as input, we replace the encoder with a small Transformer, add the intra-block position embedding, and add the three auxiliary uncertainty features (entropy $\tilde{H}$, top-1/top-2 gap Δ, and mask ratio ρ). The results in the table empirically demonstrate the benefit of using a Transformer architecture over an MLP, and show that all components contribute to maximizing the TPS while preserving the task accuracy.

Table 3: Ablation of LPS components on LLaDA-8B-Instruct and GSM8K-CoT.

Variant	**Acc.(%)**	**TPS**
LPS Router in MLP	78.39	87.40
+ MLP → Transformer	78.25	97.31
+ Intra-block position	77.94	104.18
+ $\tilde{H}$ + Δ + ρ (ours)	78.24	**104.55**

Compatibility with Semi-AR dLLMs. Unlike the bidirectional dLLMs which forward the full sequence at every step, semi-AR dLLMs such as LLaDA 2.0 and SDAR adopt block-causal attention and compute logits only for the current active window. This does not prevent EB-Decode from applying. LBS selects the next easy-to-decode block, so its selections concentrate near the decoding frontier, which the active window usually covers. Adapting to block-causal attention only requires tightening $g_{\max}$ from 5 to 1 so that each block is filled contiguously. We evaluate on A100 GPUs with a maximum generation length of 512. Restricted to the window logits, LBS improves accuracy under both tested commit thresholds (Thr.) in all eight configurations of Tab. 4. Throughput stays close to the baseline, since LBS reuses logits the engine already computes and adds no extra forward. LPS needs no adaptation, since its input covers only the tokens inside the selected block.

Table 4: LBS on semi-AR dLLMs. HumanEval is 0-shot and GSM8K is 5-shot.

		HumanEval		GSM8K	
Method	**Thr.**	**Acc. (%)**	**TPS**	**Acc. (%)**	**TPS**
LLaDA 2.0 (Bie et al., 2025), window 64					
Base	0.8	75.6	12.9	91.1	11.7
+ LBS (ours)	0.8	**77.4** *(+1.8)*	12.0	**91.6** *(+0.5)*	11.1
Base	0.95	79.9	9.4	91.8	8.6
+ LBS (ours)	0.95	**82.3** *(+2.4)*	8.8	**92.3** *(+0.5)*	8.2
SDAR-1.7B-Chat-b32 (Cheng et al., 2025), window 32					
Base	0.8	42.1	38.4	68.5	39.3
+ LBS (ours)	0.8	**43.9** *(+1.8)*	33.8	**69.5** *(+1.0)*	35.8
Base	0.9	46.3	34.2	73.1	32.8
+ LBS (ours)	0.9	**47.6** *(+1.3)*	30.2	**73.6** *(+0.5)*	29.9

Performance on Different Generation Lengths. Tab. 5 shows results of applying EB-Decode to LLaDA-8B-Instruct on GSM8K-CoT at generation lengths $L \in \{256, 512\}$ on H100 GPUs. EB-Decode achieves the highest throughput and comparable accuracy at both lengths. Its speedup also grows with generation length, suggesting that EB-Decode remains especially effective for long-sequence generation.

Table 5: Results at different gen. lengths.

	L=256		L=512	
Method	**Acc. (%)**	**Sp.up**	**Acc. (%)**	**Sp.up**
Vanilla	75.13	1.00×	79.07	1.00×
LBS	74.98	3.27×	79.68	5.09×
LBS+LPS	74.60	**3.86**×	78.24	**6.14**×

Effect of KV-Cache Configuration. Tab. 6 evaluates EB-Decode with different KV-cache strategies on LLaDA-8B-Instruct and GSM8K-CoT, measured on 4×A100 GPUs. Since LBS uses non-contiguous active blocks, we apply caching at block boundaries: prefix cache (Wu et al., 2025) reuses keys/values for tokens left of the leftmost selected position, and dual cache (Wu et al., 2025) additionally reuses those right of the rightmost selected position. Both variants further improve EB-Decode's throughput while preserving accuracy, demonstrating that it integrates with standard KV-cache pipelines for additional speedup. With KV caching applied to both methods, EB-Decode outperforms Fast-dLLM by 1.28×, 1.22×, and 1.38× with no cache, prefix cache, and dual cache, respectively, showing that its advantage holds even when the baseline also uses KV caching.

Table 6: Accuracy and TPS comparisons under different cache strategies.

Method ($L = 512$)	**Acc. (%)**	**TPS**
Fast-dLLM (Wu et al., 2025)	78.70	50.90
Fast-dLLM (Prefix Cache)	78.92	62.03
Fast-dLLM (Dual Cache)	78.70	56.92
EB-Decode	**78.24**	65.27
EB-Decode (Prefix Cache)	77.93	75.95
EB-Decode (Dual Cache)	77.93	**78.43**

Overhead of LBS and LPS Routers. We measure the wall-clock overhead of the two routers as a fraction of the per-step backbone forward time on A100. LBS accounts for roughly 3.80% and LPS for 2.41% of the per-step latency (6.21% in total), confirming that the speedups come from jointly decoding low-entropy tokens earlier before they reach the confidence threshold and thereby using fewer decoding steps, while the routers themselves introduce only negligible inference overhead.

Generality and Broader Comparisons. In Appendix G, the routers transfer without retraining across same-series models, generation lengths, and from math to code. Under the same `transformers` backend, EB-Decode on LLaDA-8B-Instruct is about 2.4× faster than LLaMA3-8B-Instruct.

6 Conclusion

dLLMs promise parallel decoding but suffer from inefficient inference under fixed block sizes and confidence thresholds. To understand this inefficiency, we identify two empirical phenomena: a staircase entropy pattern across token positions and early convergence across denoising steps. Building on these observations, we propose EB-Decode, a plug-in framework that replaces the fixed block size and confidence threshold with two lightweight learnable routers, LBS and LPS, while keeping the base dLLM frozen. Experiments on 3 dLLMs and 4 benchmarks show that EB-Decode achieves up to 18.76× higher throughput than the vanilla decoder with comparable accuracy, a practical step toward narrowing the efficiency gap between dLLMs and autoregressive models.

AI USE STATEMENT

We used generative AI tools for polishing the writing of this paper and for assisting with the implementation of our experimental code. All AI-assisted text and code were reviewed and verified by the authors, and we take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure that our results are reproducible. Sec. 5.1 lists the base models, benchmarks, training data, and router training settings, all of which rely on publicly available models and datasets. Appendices C and D describe the architectures, training procedures, and training dynamics of the LBS and LPS routers, and Appendix E gives the full inference procedure. Appendix F reports the decoding configuration, all EB-Decode hyperparameters, and how throughput is measured. We will release the source code upon acceptance.

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. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

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

Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. *arXiv preprint arXiv:2509.25188*, 2025.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Ling Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b, 2025. URL https://arxiv.org/abs/2512.15745.

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

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 2195–2207, 2021b.

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai Li, Yiran Chen, et al. Dpad: Efficient diffusion language models with suffix dropout. *arXiv preprint arXiv:2508.14148*, 2025a.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488*, 2025b.

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

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

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Justin Deschenaux and Caglar Gulcehre. Beyond autoregression: Fast llms via self-distillation through time. *arXiv preprint arXiv:2410.21035*, 2024.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Google DeepMind. Gemini diffusion. https://deepmind.google/models/gemini-diffusion/, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Naibin Gu, Peng Fu, Xiyu Liu, Bowen Shen, Zheng Lin, and Weiping Wang. Light-PEFT: Lightening parameter-efficient fine-tuning via early pruning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7528–7541, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.447. URL https://aclanthology.org/2024.findings-acl.447/.

Satoshi Hayakawa, Yuhta Takida, Masaaki Imaizumi, Hiromi Wakaki, and Yuki Mitsufuji. Distillation of discrete diffusion through dimensional correlations. *arXiv preprint arXiv:2410.08709*, 2024.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578*, 2025.

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

Pengcheng Huang, Shuhao Liu, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, and Tong Xiao. Pc-sampler: Position-aware calibration of decoding bias in masked diffusion models. *arXiv preprint arXiv:2508.13021*, 2025.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv e-prints*, pp. arXiv–2506, 2025.

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

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf, 2024.

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. *arXiv preprint arXiv:2508.19982*, 2025.

Pengxiang Li, Dilxat Muhtar, Tianlong Chen, Lu Yin, and Shiwei Liu. Why diffusion language models struggle with truly parallel (non-autoregressive) decoding? *arXiv preprint arXiv:2602.23225*, 2026.

Yihao Liang, Ze Wang, Hao Chen, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Emad Barsoum, Zicheng Liu, and Niraj K Jha. Cd4lm: Consistency distillation and adaptive decoding for diffusion language models. *arXiv preprint arXiv:2601.02236*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 158–167, 2017.

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

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

Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size. *arXiv preprint arXiv:2509.26432*, 2025.

Lizhuo Luo, Shenggui Li, Yonggang Wen, and Tianwei Zhang. Dsb: Dynamic sliding block scheduling for diffusion llms. *arXiv preprint arXiv:2602.05992*, 2026.

Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed–dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025a.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025b.

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

Han Peng, Peiyu Liu, Zican Dong, Daixuan Cheng, Junyi Li, Yiru Tang, Shuo Wang, and Wayne Xin Zhao. How efficient are diffusion language models? a critical examination of efficiency evaluation practices. *arXiv preprint arXiv:2510.18480*, 2025.

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

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025a.

Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. Time is a feature: Exploiting temporal dynamics in diffusion language models. *arXiv preprint arXiv:2508.09138*, 2025b.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025c.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with slowfast sampling: The three golden principles. *arXiv preprint arXiv:2506.10848*, 2025.

Lexington Whalen, Zhenbang Du, Haoran You, Chaojian Li, Sixu Li, and Yingyan Celine Lin. Early-bird diffusion: Investigating and leveraging timestep-aware early-bird tickets in diffusion models for efficient training. *arXiv preprint arXiv:2504.09606*, 2025.

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

Kejing Xia, Mingzhe Li, Lixuan Wei, Zhenbang Du, Xiangchi Yuan, Dachuan Shi, Qirui Jin, and Wenke Lee. Metastate: Persistent working memory enhances reasoning in discrete diffusion language models. *arXiv preprint arXiv:2603.01331*, 2026.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 2246–2251, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Celine Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

Haoran You, Zhihan Lu, Zijian Zhou, Yonggan Fu, and Yingyan Lin. Early-bird gcns: Graph-network co-optimization towards more efficient gcn training and inference via drawing early-bird lottery tickets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8910–8918, 2022.

Tunyu Zhang, Xinxi Zhang, Ligong Han, Haizhou Shi, Xiaoxiao He, Zhuowei Li, Hao Wang, Kai Xu, Akash Srivastava, Vladimir Pavlovic, et al. T3d: Few-step diffusion language models via trajectory self-distillation with direct discriminative optimization. *arXiv preprint arXiv:2602.12262*, 2026a.

Yu Zhang, Xinchen Li, Jialei Zhou, Hongnan Ma, Zhongwei Wan, Yiwei Shi, Duoqian Miao, Qi Zhang, and Longbing Cao. Swordsman: Entropy-driven adaptive block partition for efficient diffusion language models. *arXiv preprint arXiv:2602.04399*, 2026b.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *International Conference on Learning Representations*, 2025.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33: 18330–18341, 2020.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025a.

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

A Related Works on dLLMs

Unlike AR models, dLLMs generate text through an iterative denoising process over discrete tokens (Austin et al., 2021a; Sahoo et al., 2024; Shi et al., 2024; Lou et al., 2023; Zheng et al., 2025), achieving likelihood comparable to their AR counterparts. D3PM (Austin et al., 2021a) generalizes discrete diffusion using structured transition matrices over the token vocabulary. Subsequent works refine the masked diffusion paradigm. MDLM (Sahoo et al., 2024) derives a Rao-Blackwellized objective that reduces to a weighted mixture of masked language modeling losses. MD4 (Shi et al., 2024) further simplifies and generalizes the continuous-time objective. SEDD (Lou et al., 2023) proposes a score-entropy objective. Zheng et al. (2025) shows that masked diffusion training and sampling are time-agnostic. Together, these advances bring masked dLLMs to likelihood performance comparable to AR models. LLaDA (Nie et al., 2025) performs on par with LLaMA3 at the billion-parameter scale. Subsequent extensions add variance-reduced preference optimization (Zhu et al., 2025a), sparse experts (Zhu et al., 2025b), and AR-to-diffusion conversion that scales to 100B parameters (Bie et al., 2025). Dream-7B (Ye et al., 2025) introduces AR-based LLM initialization and context-adaptive, token-level noise rescheduling. Semi-AR methods such as BD3-LMs (Arriola et al., 2025) and SDAR (Cheng et al., 2025) combine the coherence of AR generation with the parallelism of diffusion. They maintain AR dependencies across blocks while sampling tokens in parallel within each block. Beyond decoding efficiency, MetaState (Xia et al., 2026) augments a frozen dLLM with a lightweight recurrent working memory that carries information across denoising steps, improving reasoning.

B Preliminaries of dLLMs

dLLMs formulate text generation as a diffusion process over discrete token sequences, consisting of a forward masking process and a reverse denoising process. Let $\mathbf{x}_0 = (x_0^1, \ldots, x_0^L)$ denote a clean token sequence of length L drawn from a vocabulary $\mathcal{V}$, where x_0^i is the token at position i. The forward process samples a noise level $t \in [0, 1]$ and produces a corrupted sequence $\mathbf{x}_t = (x_t^1, \ldots, x_t^L)$ by independently replacing each token with the special `[MASK]` symbol with probability t, so that $\mathbf{x}_t$ converges to a fully masked sequence as $t \to 1$. The reverse process is parameterized by a bidirectional Transformer that defines a mask predictor $p_\theta(\cdot \mid \mathbf{x}_t)$, which recovers the distribution over the original token at every masked position simultaneously. The model is trained by minimizing

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\,\mathbf{x}_0,\,\mathbf{x}_t}\left[\frac{1}{t}\sum_{i=1}^{L}\mathbf{1}_{\{x_t^i=\texttt{[MASK]}\}}\big(-\log p_\theta(x_0^i \mid \mathbf{x}_t)\big)\right], \tag{4}$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function that restricts the cross-entropy to currently masked positions, and the factor $1/t$ compensates for the expected mask ratio at noise level t. This objective upper-bounds the negative log-likelihood of the model distribution.

Inference-time denoising trajectory. At inference time, given a prompt c, decoding starts from a fully masked response sequence and iteratively applies the reverse process from $t = 1$ to $t = 0$ over a sequence of denoising steps. The prompt c is concatenated with the response and processed jointly by the bidirectional Transformer (Nie et al., 2025), so that the predictive distribution at every masked position conditions on both $\mathbf{x}_t$ and c. The final output is the fully unmasked sequence $\mathbf{x}_0$.

Block partition and in-block prediction. The response is divided into contiguous blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots$ of fixed size B, indexed by j, and processed from left to right. Within the active block $\mathcal{B}_j$, the model predicts all masked positions in parallel, commits a subset of tokens, and remasks the remaining uncertain ones for further refinement. The per-position top-1 confidence at masked position $i \in \mathcal{B}_j$ over candidate vocabulary item $v \in \mathcal{V}$ is

$$\kappa^i = \max_{v\in\mathcal{V}} p_\theta\big(x_0^i = v \;\big|\; \mathbf{x}_t,\, c\big). \tag{5}$$

Given a confidence threshold τ or an optional per-step commit budget m, the standard block update rule is

$$x_t^i \leftarrow \begin{cases} \arg\max\limits_{v\in\mathcal{V}} p_\theta\big(x_0^i = v \;\big|\; \mathbf{x}_t,\, c\big), & \kappa^i \ge \tau \;\text{ or }\; i \in \text{Top-}m(\mathcal{B}_j), \\ \texttt{[MASK]}, & \text{otherwise}, \end{cases} \tag{6}$$

where Top-$m(\mathcal{B}_j)$ denotes the m masked positions in $\mathcal{B}_j$ with the highest top-1 confidence at the current step. Once all positions in $\mathcal{B}_j$ have been committed, the active block advances to $\mathcal{B}_{j+1}$ and the same procedure repeats. Throughout this recipe, the block size B, the threshold τ, and the budget m remain fixed regardless of how token difficulty varies along the sequence, which is precisely the limitation that motivates our EB-Decode framework. Building on the same block-wise procedure and keeping the base dLLM frozen, EB-Decode replaces the two static design choices with learnable counterparts. Specifically, LBS replaces the fixed block size B with a learnable block selector as described in Sec. 4.1, while LPS replaces the static commit rule with a position-aware, learnable acceptance strategy as described in Sec. 4.2.

C Detailed Setup for Learnable Block Size

C.1 Router Architecture

The LBS router $\mathcal{R}_\phi$ is a small Transformer encoder with 600k parameters (not including the frozen token embedding table W_e) that maps per-masked-position features to scalar inclusion logits.

Entropy branch. The normalized entropy sequence for all masked tokens $\tilde{H} = \{\tilde{H}^i : i \in \mathcal{M}\}$ is first zero-padded to the generation length L and then processed by a two-layer MLP F_{ent}: Linear$(1 \to 64) \to$ GELU $\to$ Linear$(64 \to 64) \to$ LayerNorm, producing a 64-dimensional entropy embedding.

Token branch. The top-1 predicted token IDs $\hat{x} = \{\hat{x}^i : i \in \mathcal{M}\}$ are looked up in the frozen base-model token embedding table W_e, zero-padded to L, and projected to $\mathbb{R}^{64}$ via a learned linear layer $W_{\text{tok}} \in \mathbb{R}^{64 \times d_{\text{base}}}$.

Fusion and positional embedding. The two 64-dimensional embeddings are concatenated to form a 128-dimensional vector, then passed through a fusion layer $W_{\text{combine}} \in \mathbb{R}^{128 \times 128}$. The final input embedding to the Transformer encoder is calculated as:

$$Z_\phi = W_{\text{combine}} \Big[F_{\text{ent}}\Big(\tilde{H}\Big) \;;\; W_{\text{tok}}\, W_e\big(\hat{x}\big) \Big] + E_{\text{pos}}, Z_\phi \in \mathbb{R}^{L \times 128} \tag{7}$$

where $E_{\text{pos}} \in \mathbb{R}^{L \times 128}$ is a learnable positional embedding.

Transformer encoder. The sequence Z_ϕ is then fed into a 2-layer Transformer encoder with hidden dimension d_ϕ=128, 4 attention heads, FFN width 256, and GELU activations.

Output head. A linear head $\mathbb{R}^{128} \to \mathbb{R}^1$ maps the encoder output to logits $O_\phi \in \mathbb{R}^{L \times 1}$. The padding is removed before returning, yielding an output of shape $(|\mathcal{M}|, 1)$. The final inclusion probability at position i is $p_\phi^i = \sigma(O_\phi^i)$.

C.2 Training Procedure

The full per-step training procedure for the LBS router, summarized in the main text and the simplified Alg. 1, is given in Alg. 3. Training is performed on NVIDIA H100 GPUs.

Training stability. Tab. 7 reports the training and validation losses of the LBS router. The training loss is negative because Eq. (2) subtracts the regularization term from the weighted cross-entropy loss. Both losses drop sharply in the first epoch, after which the changes are much smaller, indicating stable training.

Table 7: Training and validation losses of the LBS router across epochs.

Epoch	Train loss	Val loss
0	0.0569	0.0764
1	−0.337	−0.291
3	−0.364	−0.378
6	−0.359	−0.396

Table 8: Ablation of LBS components on LLaDA-8B-Instruct and GSM8K-CoT. Each row progressively adds one component.

Variant	Acc. (%)	TPS
LBS Router	74.53	42.25
+ EOS removal	77.33	90.31
+ Max-gap constraint	78.70	86.92
+ Min-tokens constraint	79.68	86.80

Algorithm 3 LBS Router Training

Require: dLLM-generated response dataset $\mathcal{D}$, frozen dLLM $\mathcal{F}_\theta$, router $\mathcal{R}_\phi$, generation length L, block length L_B, mask ratio $r \in (0, 1]$, learning rate η_{lr}

Notation:

$\tilde{H}(p) = (-\sum_{v \in \mathcal{V}} p(v) \log p(v)) / \log |\mathcal{V}|$

$\hat{x}(p) = \arg\max_x p(x)$

$\sigma(p) = 1/(1 + e^{-p})$

for $x \in \mathcal{D}$ **do**
  Sample $n \sim \{0, 1, \ldots, L/L_B - 1\}$
  $x_{\text{masked}} \leftarrow$ Blockwise-Masking(x, r, n, L, L_B); $\mathcal{M} \leftarrow \{i : x^i_{\text{masked}} = \texttt{[MASK]}\}$
  for each $i \in \mathcal{M}$ **do**
    $p^i \leftarrow \text{softmax}(\mathcal{F}_\theta(x_{\text{masked}})^i)$
    $\ell^i_{\text{CE}} = \text{CrossEntropy}(p^i, x^i)$ ▷ per-token CE against ground truth
    $O^i_\phi \leftarrow \mathcal{R}_\phi(\tilde{H}(p^i), \hat{x}(p^i))$
  end for
  $\mathcal{L}_{\text{wCE}} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \sigma(O^i_\phi) \cdot \ell^i_{\text{CE}}$
  $\mathcal{L}_{\text{Reg}} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \sigma(O^i_\phi)$
  $\mathcal{L}_{\text{LBS}} \leftarrow \mathcal{L}_{\text{wCE}} - \mathcal{L}_{\text{Reg}}$
  $\mathcal{R}_\phi \leftarrow \mathcal{R}_\phi - \eta_{\text{lr}} \nabla_\phi \mathcal{L}_{\text{LBS}}$ ▷ update only router; $\mathcal{F}_\theta$ is frozen
end for
return $\mathcal{R}_\phi$

C.3 Ablation of LBS Filters

Tab. 8 additively builds up the LBS filter described in Sec. 4.1 on LLaDA-8B-Instruct and GSM8K-CoT. Starting from the LBS router's selected block, we incrementally add EOS removal, the max-gap constraint, and the min-tokens constraint to further refine the selected block. The results show that all filters are crucial in maximizing the task accuracy while maintaining high TPS.

D Detailed Setup for Learnable Parallel Sampling

D.1 Router Architecture

Input features. Besides the top-1 confidence κ^i_t, the router uses three auxiliary features. The normalized entropy $\tilde{H}^i_t$ measures absolute prediction uncertainty, but alone it cannot tell whether the top-1 candidate dominates. The confidence gap Δ^i_t between the top-1 and top-2 predictions captures this relative ambiguity, namely the margin over the strongest competitor. The block-wise mask ratio ρ_t reflects global decoding progress, since the commit policy should differ across denoising stages.

The LPS router $\mathcal{R}_\psi$ is a small Transformer encoder that operates over all positions of the active block jointly. Taking the per-position input (f^i, pos^i) defined in Sec. 4.2, the input embedding Z_ψ is a linear projection followed by LayerNorm and GELU, mapping each feature vector into a hidden space of dimension $d_\psi = 32$, so that $Z_\psi \in \mathbb{R}^{|\mathcal{B}| \times 32}$. The encoder then processes Z_ψ with 2 pre-norm Transformer blocks with 4 attention heads, a feed-forward width of 64, GELU activations, and dropout of 0.1. A final linear head projects each hidden state to a scalar logit O^i_ψ, from which the commit probability is $p^i_\psi = \sigma(O^i_\psi)$, giving a total of roughly 27k trainable parameters. This keeps the router cost negligible compared with the base dLLM forward pass.

D.2 Trace Generation

To remove any train/test distribution skew, we collect traces under the exact decoding configuration that the router will face at inference. For each experiment we run the corresponding frozen base dLLM with generation length 512, base block size 32, and confidence threshold τ=0.9, matching the inference setting in Sec. 5.1. Since the distribution differs across base models, we train a separate router for each base model. Every record captures, at each position i of the active block,

the four feature components $f_t^i = (\kappa_t^i, \tilde{H}_t^i, \Delta_t^i, \rho_t)$, the predicted token $\hat{x}_t^i$, the correctness label $y_t^i = \mathbf{1}[\hat{x}_t^i = x_0^i]$ obtained by comparing against the completed sequence x_0, and the mask indicator m_t^i which shows which position is still masked. The replay pass follows the oracle progress rule described in Sec. 4.2, so each intermediate state stays consistent with x_0 and the labels y_t^i provide a clean supervision signal without external annotation.

Algorithm 4 LPS Router Training

Require: Frozen dLLM $\mathcal{F}_\theta$; prompt corpus $\mathcal{D}$; LBS Router $\mathcal{R}_\phi$; LPS Router $\mathcal{R}_\psi$; negative weight $w^- = 2$; target False-Positive rate $\eta \in (0, 1)$; learning rate η_{lr}; early-stop patience P_{patience}.
Ensure: Router parameters $\mathcal{R}_\psi$ and acceptance threshold τ_ψ.

Stage 1: build the trace dataset $\mathcal{T}$.

$\mathcal{T} \leftarrow \emptyset$ ▷ $\mathcal{T}$ stores per-position records $(f_t^i,\ \text{pos}^i,\ y_t^i)$.
for all prompt $q \in \mathcal{D}$ **do** ▷ Run LBS decoding on q to obtain x_0 and the per-step trajectory.
  for all step t and masked position $i \in \mathcal{B}$ at step t **do**
    $p_t^i \leftarrow \text{softmax}\big(\mathcal{F}_\theta(\mathbf{x}_t)\big)_i$ ▷ predictive distribution at position i from $\mathbf{x}_t$.
    $\kappa_t^i \leftarrow \max_v p_t^i(v),\ \hat{x}_t^i \leftarrow \arg\max_v p_t^i(v)$ ▷ top-1 confidence and predicted token.
    $\tilde{H}_t^i,\ \Delta_t^i,\ \rho_t \leftarrow$ entropy, top-1/top-2 gap, block-wise mask ratio
    $f_t^i \leftarrow (\kappa_t^i,\ \tilde{H}_t^i,\ \Delta_t^i,\ \rho_t),\ \text{pos}^i \leftarrow {}^{(i\,-\,b)}/_{|\mathcal{B}|}$
    $y_t^i \leftarrow \mathbf{1}[\hat{x}_t^i = x_0^i]$ ▷ correctness label vs. x_0.
    Append $(f_t^i,\ \text{pos}^i,\ y_t^i)$ to $\mathcal{T}$
  end for
end for

Stage 2: LPS router $\mathcal{R}_\psi$ training.

repeat
  for all mini-batch $\mathcal{S} \subset \mathcal{T}_{\text{train}}$ **do**
    $O_\psi^i \leftarrow \mathcal{R}_\psi(f_t^i,\ \text{pos}_t^i)$ for each record $(f_t^i,\ \text{pos}^i,\ y_t^i) \in \mathcal{S}$
    $\mathcal{L}_{\text{LPS}} \leftarrow \dfrac{1}{|\mathcal{S}|} \displaystyle\sum_{(i,t)\in\mathcal{S}} w_t^i\, \text{BCE}(\sigma(O_\psi^i),\ y_t^i), \quad \text{where } w_t^i = \begin{cases} 1, & y_t^i = 1 \\ w^-, & y_t^i = 0 \end{cases}$ ▷ mini-batch form of Eq. (3)
    $\mathcal{R}_\psi \leftarrow \mathcal{R}_\psi - \eta_{\text{lr}}\, \nabla \mathcal{L}_{\text{LPS}}$
  end for
  Evaluate validation recall on $\mathcal{T}_{\text{val}}$ under $\text{FP} \le \eta$
until validation recall has not improved for P_{patience} consecutive epochs
get $\mathcal{T}_{\text{val}}$ s.t. $|\text{FP}_{\mathcal{T}_{\text{val}}}(\tau_\psi) - \eta| \le \delta$ ▷ within $\eta \pm 0.5$ pp false-positive rate on validation set.
return $(\mathcal{R}_\psi,\ \tau_\psi)$

D.3 Training Procedure

Valid trajectories are split at the trace level into 80% training, 10% validation, and 10% test sets, with the split deterministic under seed 42 so that every router variant sees identical data. The positions outside the active block are masked out of both the loss and the attention computation to prevent boundary artifacts from contaminating the supervision.

Optimization uses AdamW with an initial learning rate of $1\text{e}{-3}$, weight decay 0.01, gradient clipping at norm 1.0, and automatic mixed precision. The schedule consists of a linear warmup over 0.5% of the total steps followed by a per-epoch geometric decay with ratio 0.97. We train with a batch size of 512 step records for up to 2000 epochs, with early stopping triggered when the validation recall at the false-positive budget fails to improve for 100 consecutive epochs. Training is performed on NVIDIA H100 GPUs.

The training objective is the masked weighted binary cross-entropy defined in Eq. (3), where negative samples carry a weight of $w^- = 2$. We use asymmetric weights because a false positive commits a wrong token irreversibly, while a false negative only delays acceptance to a later step. Premature acceptance therefore has a higher cost.

We tune the LPS threshold τ_ψ on the validation set at the end of each improving epoch by searching over a 250-point grid in $[0.8,\ 0.9999]$ and keeping the largest recall under a false-positive-rate budget of $\eta = 10^{-2}$. The full procedure is summarized in Alg. 4.

Training stability. As shown in Tab. 9, the training and validation losses converge smoothly and the validation recall rises steadily. The validation loss stays close to the training loss throughout, indicating no overfitting.

Table 9: Training and validation losses and validation recall of the LPS router across epochs.

Epoch	Train loss	Val loss	Val recall
0	0.6225	0.3987	0.6471
20	0.3699	0.3640	0.6998
60	0.3677	0.3622	0.7042
119	0.3669	0.3618	**0.7052**

E EB-DECODE INFERENCE PROCEDURE

The complete inference procedure that combines LBS with LPS is given in Alg. 5. The simplified high-level view is in Alg. 2.

F EXPERIMENTAL SETUP

Decoding configuration. All methods share the same inference configuration of generation length 512, base block size 32, and base confidence threshold τ=0.9. No method in Tab. 1 uses a KV cache. In the official implementation of Fast-dLLM (Wu et al., 2025), the KV cache is recomputed with one full-sequence forward pass at the start of each block and reused within that block, whereas the vanilla LLaDA decoding loop maintains no cache.

EB-Decode hyperparameters. For LBS in our EB-Decode, we set $g_{\max} = 5$ for all three dLLMs, and $L_{\min} = 8, 8, 16$ for LLaDA-8B-Instruct, Dream-v0-Instruct-7B, LLaDA-1.5 respectively. When not paired with LPS, LBS adopts confidence-based decoding following Fast-dLLM (Wu et al., 2025). For LPS, we choose the acceptance threshold τ_ψ under a false-positive-rate budget of $1\% \pm 0.5$ % points on the validation set, giving $\tau_\psi = 0.89$, 0.90, and 0.87 for LLaDA-8B-Instruct, Dream-v0-Instruct-7B, LLaDA-1.5 respectively.

Throughput measurement. All methods, including every baseline, are measured with the HuggingFace `transformers` backend at batch size 1, following prior dLLM acceleration works (Lu et al., 2025; Bao et al., 2025; Chen et al., 2025b), and throughput (TPS) is measured per GPU as the number of generated tokens divided by the end-to-end latency.

G GENERALITY AND BROADER COMPARISONS

Router Transferability. The routers of EB-Decode are largely transferable, so they can be reused without retraining when the deployment setting changes. We examine this transferability from three aspects: across models, across generation lengths, and across tasks, with all runs on NVIDIA A100 GPUs. The experiments in Tabs. 10 and 11 were run on a different A100 server from those in Tab. 6, so their accuracy and TPS differ slightly from the corresponding entries there (e.g., 78.17% and 69.54 TPS vs. 78.24% and 65.27 TPS for LLaDA-8B-Instruct at L=512). (1) Across models as shown in Tab. 10 (a), we consider two transfer targets. For LLaDA-1.5, which belongs to the same model series as LLaDA-8B-Instruct, transferring LBS alone or together with LPS matches the natively trained routers in accuracy and throughput. For Dream-v0-Instruct-7B, which comes from a different model series, LBS does not transfer, because it embeds predicted token IDs with the base model's embedding table and vocabularies differ across series. LPS, whose features are vocabulary-independent, still transfers. (2) Across generation lengths as shown in Tab. 10 (b), an LPS router trained at one length remains on par at the other. Only LPS is evaluated here, because LBS takes the entire masked sequence as input and uses positional embeddings tied to the generation length (Appendix C.1). We therefore recommend using LBS at its training length. (3) Across tasks, both routers are trained only on mathematical data, so the HumanEval and MBPP results in Tab. 1 already reflect transfer to unseen code tasks.

Comparison with Autoregressive Models. To examine whether EB-Decode narrows the efficiency gap between dLLMs and autoregressive (AR) models, Tab. 11 compares it with Qwen3-8B (Yang

Algorithm 5 Inference with LBS Block Selection and LPS Parallel Commit

Require: Prompt p, frozen dLLM $\mathcal{F}_\theta$; LBS router $\mathcal{R}_\phi$; LPS router $\mathcal{R}_\psi$; generation length L; min block size $L_{\min}$; fallback block size L_B; max gap $g_{\max}$; confidence threshold τ; LPS acceptance threshold τ_ψ

1: $x \leftarrow [p \mid \underbrace{\texttt{[MASK]}, \ldots, \texttt{[MASK]}}_{L}]$; $\mathcal{B} \leftarrow \emptyset$
2: **while** $\exists\, i$ such that $x^i = \texttt{[MASK]}$ **do**
3: $p_t \leftarrow \mathrm{softmax}\big(\mathcal{F}_\theta(x)\big)$; $\mathcal{M} \leftarrow \{i : x^i = \texttt{[MASK]}\}$ ▷ get distribution and masked places
4: **for** each $i \in \mathcal{M}$ **do**
5: $\kappa^i \leftarrow \max p_t^i$; $\hat{x}^i \leftarrow \arg\max p_t^i$
6: Compute $\tilde{H}^i$, Δ^i, ρ; ▷ compute entropy, top-1/top-2 gap, block-wise mask ratio
7: $f^i \leftarrow (\kappa^i,\, \tilde{H}^i,\, \Delta^i,\, \rho)$ ▷ LPS feature tuple
8: **end for**
9: **if** $\mathcal{B} = \emptyset$ **then** ▷ LBS: select a new active block
10: **if** $|\mathcal{M}| \le L_{\min}$ **then**
11: $\mathcal{B} \leftarrow \mathcal{M}$ ▷ size fallback
12: **else**
13: $\mathcal{C} \leftarrow \{i \in \mathcal{M} : \sigma(\mathcal{R}_\phi(\tilde{H}^i,\, \hat{x}^i)) > 0.5 \ \wedge\ \hat{x}^i \neq \mathrm{EOS}\}$ ▷ router + EOS removal
14: $\mathcal{B} \leftarrow \mathrm{MaxGapFilter}(\mathcal{C},\, g_{\max})$ ▷ spatial coherence
15: **if** $|\mathcal{B}| < L_{\min}$ **then**
16: $\mathcal{B} \leftarrow$ first L_B positions of $\mathcal{M}$ ▷ min-tokens fallback
17: **end if**
18: **end if**
19: **end if**
20: $O_\psi^i \leftarrow \mathcal{R}_\psi(f^i,\, {}^{(i-b)}/_{|\mathcal{B}|})$ for $i \in \mathcal{B}$ ▷ LPS router, b is the left boundary index of $\mathcal{B}$
21: $\mathcal{A} \leftarrow \big\{i \in \mathcal{B} : x^i = \texttt{[MASK]} \ \wedge\ \big(\kappa^i \ge \tau \ \vee\ p_\psi^i > \tau_\psi\big)\big\}$ ▷ union accept rule
22: **if** $\mathcal{A} = \emptyset$ **then**
23: $\mathcal{A} \leftarrow \{\arg\max_{i \in \mathcal{B},\, x^i = \texttt{[MASK]}} \kappa^i\}$ ▷ anti-stall fallback
24: **end if**
25: $x^i \leftarrow \hat{x}^i$ for $i \in \mathcal{A}$ ▷ parallel commit
26: **if** $x^i \neq \texttt{[MASK]}$ for all $i \in \mathcal{B}$ **then**
27: $\mathcal{B} \leftarrow \emptyset$ ▷ If $\mathcal{B}$ resolved, move to Line 9
28: **end if**
29: **end while**
30: **return** x

Table 10: Router transfer without retraining on GSM8K-CoT, (a) across models and (b) across generation lengths. "Native" uses routers trained in the target setting, and "Transferred" uses the router(s) listed in the second column.

		Native		Transferred	
Target	**Transferred router**	**Acc. (%)**	**TPS**	**Acc. (%)**	**TPS**
(a) Across models					
LLaDA-1.5	LBS from LLaDA-8B	79.23	64.95	79.15	64.68
LLaDA-1.5	LBS + LPS from LLaDA-8B	79.23	64.95	79.23	66.37
Dream-v0-Instruct-7B	LPS from LLaDA-8B	76.19	43.55	76.49	43.07
(b) Across generation lengths (LLaDA-8B-Instruct)					
L=512	LPS trained at L=256	78.17	69.54	78.24	66.95
L=256	LPS trained at L=512	74.68	95.37	74.15	94.08

et al., 2025) and LLaMA3-8B-Instruct (Grattafiori et al., 2024), with all models run on the same `transformers` backend with batch size 1 on 4×A100 GPUs. The low Acc. of the AR models reflects a format mismatch with the standard answer-extraction rule of the GSM8K harness, while Flex. Acc. extracts the answer from `\boxed{}` or the last number in the response and is thus robust to the output format. While vanilla dLLM decoding is less than half as fast as either AR model, EB-Decode runs about 2.4× faster than the faster of the two, and the remaining accuracy gap to

Table 11: Comparison with AR models on GSM8K-CoT (L=512). Acc. follows the standard answer extraction of the GSM8K harness, while Flex. Acc. takes the content of `\boxed{}` or the last number in the response and is thus robust to the output format.

Model	Type	TPS	Acc. (%)	Flex. Acc. (%)
Qwen3-8B (Yang et al., 2025)	AR	21.22	32.22	**91.89**
LLaMA3-8B-Instruct (Grattafiori et al., 2024)	AR	28.94	68.54	80.36
LLaDA-8B-Instruct, Vanilla (Nie et al., 2025)	dLLM	10.24	**79.07**	84.08
LLaDA-8B-Instruct, EB-Decode	dLLM	**69.54**	78.17	82.49

Qwen3-8B in Flex. Acc. stems mostly from the base model rather than the decoding method. Beyond this matched setting, AR models benefit from recent serving systems such as vLLM (Kwon et al., 2023), where Qwen3-8B and LLaMA3-8B-Instruct reach 5689.70 and 4965.19 TPS through paged KV-cache memory management, continuous batching, and optimized attention kernels (Dao et al., 2022), whereas serving frameworks for dLLMs such as dInfer (Ma et al., 2025b) are still in early development.

H Positioning Against Prior Adaptive dLLM Decoding Methods

To make explicit how EB-Decode differs from prior adaptive dLLM decoding methods, Tab. 12 compares them along two axes: *where to decode*, i.e., how the block is formed, and *when to commit*, i.e., how tokens are accepted. Prior methods either fix the block size or form contiguous blocks with hand-designed heuristics, and most of them commit tokens with a static confidence threshold. EB-Decode is the only one that learns both axes and supports non-contiguous, variable-length blocks, so hard tokens can be deferred and resolved with richer right-side context once the surrounding easy tokens are decoded (Fig. 1).

Table 12: Positioning against prior dLLM decoding methods along the two decoding axes.

Method	Where to decode	When to commit	Non-contiguous blocks
Fast-dLLM (Wu et al., 2025)	fixed size	confidence threshold	no
AdaBlock-dLLM (Lu et al., 2025)	delimiter heuristic	confidence threshold	no
Swordsman (Zhang et al., 2026b)	entropy heuristic	confidence threshold	no
Learn2PD (Bao et al., 2025)	fixed size	learned (MLP, fixed length)	no
EB-Decode (ours)	**learned**	**learned (Transformer, variable length)**	**yes**

I Entropy at the Sequence Tail

Fig. 2 (a) shows a persistent band of low entropy at high token positions across all decoding steps, which may appear at odds with the overall rising-entropy trend. This region corresponds to the EOS zone at the tail of the sequence. We report in Tab. 13 the three most frequent predictions at the last five positions across all decoding steps, where `<|endoftext|>` is overwhelmingly dominant, appearing in the top-5 predictions at 96.30% of the positions sampled. These positions therefore remain highly certain at almost every step, forming the persistent low-entropy band.

Table 13: Top-3 predicted tokens at the last 5 positions of the sequence, aggregated over all decoding steps on LLaDA-8B-Instruct. "Top-5 occurrence" is the fraction of sampled positions at which the token appears among the top-5 predictions.

Rank	Token	Avg. prob. (%)	Top-5 occurrence (%)
1	`<|endoftext|>`	28.00	96.30
2	`<|eot_id|>`	18.00	45.70
3	.	12.70	24.70

J LIMITATIONS

We outline the main limitations of EB-Decode and the directions that we leave for future work.

Generation Length. Our main results are reported at generation lengths of 256 and 512 tokens, which match the standard configurations used in prior dLLM acceleration works (Wu et al., 2025; Lu et al., 2025; Bao et al., 2025). As shown in Tab. 5, EB-Decode already exhibits a scaling trend in which the speedup grows from $L{=}256$ to $L{=}512$, suggesting that the framework remains effective as the sequence becomes longer. A more comprehensive study at substantially longer generation lengths is a natural extension we plan to pursue in future work.

Serving Conditions. Our throughput is measured at batch size 1, following prior dLLM acceleration works (Wu et al., 2025; Lu et al., 2025; Bao et al., 2025). Evaluation under realistic serving conditions with continuous batching and system-level optimizations is left to future work, as it depends as much on dLLM serving infrastructure as on the decoding algorithm.

Table 14: Throughput (TPS) of Fast-dLLM on LLaDA-8B-Instruct and GSM8K-CoT ($L{=}512$) under different cache strategies on A100 and H100. Parentheses show the speedup over the no-cache setting on the same GPU.

Cache strategy	**A100**	**H100**
No cache	50.90	79.20
Prefix cache	62.03 *(1.22×)*	82.46 *(1.04×)*
Dual cache	56.92 *(1.12×)*	71.76 *(0.91×)*

Memory-Bound Regime. Under batch size 1 and generation lengths up to 512, the full-sequence forward pass of current dLLMs is largely memory-bound, so reducing the computation per step yields limited wall-clock gains. This is why KV caching improves throughput by at most about $1.2\times$ in Tab. 6. The behavior is common to dLLM decoding rather than specific to EB-Decode. As shown in Tab. 14, the same caches give Fast-dLLM (Wu et al., 2025) a gain comparable to ours on A100, and little or no gain on H100, whose compute is higher relative to its memory bandwidth. EB-Decode reduces the number of forward passes, which is orthogonal to the per-step cost, and it outperforms Fast-dLLM under every cache configuration (Tab. 6). At longer generation lengths or on GPUs with lower compute, the forward pass may become compute-bound and caching would contribute a larger share of the speedup. We leave a systematic study of this regime to future work.

Method Complexity. EB-Decode introduces two routers and several hyperparameters, namely $g_{\max}$, $L_{\min}$, τ, and τ_ψ, which makes it more involved to deploy and reuse as a baseline than a single-threshold heuristic. Each component contributes to the final performance (Tabs. 2, 3, and 8), and the trained routers are largely reusable without retraining (Appendix G), but reducing the number of moving parts remains a worthwhile direction.