TEMPORAL DELTA ACTIVATION SPARSITY FOR DIFFUSION LANGUAGE MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models repeatedly process a sequence to generate text, making inference expensive. Existing activation-sparsity methods reduce this inference cost by discarding small activations. We instead exploit the fact that activations often change little between denoising steps. Thus, we propose *Temporal Delta Sparsity* (TDS), a training-free method that updates each linear layer using the selected changes to a stored reference input. A shared budget across tokens and channels assigns computation to the changes with the largest weight-informed scores. We show that the accumulator remains consistent with its reference in exact arithmetic and that this selection minimises a one-step output-error bound. Moreover, unprocessed changes remain in the residual for later steps. On GSM8K, MATH500, HumanEval and MBPP, TDS skips 88–89% of linear-layer multiply–accumulates. Its average score is 47.8% versus 47.4% for dense LLaDA-8B, and 59.8% versus 60.6% for dense Dream-7B. On the separate GSM8K evaluation, a whole-row variant achieves 1.90× and 1.46× end-to-end speedups, respectively, with equal or higher observed accuracy.

1 INTRODUCTION

Diffusion language models (DLMs) generate text by repeatedly predicting and revealing masked tokens (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025; Ye et al., 2025). Their bidirectional attention lets each prediction use context on both sides. However, each denoising step processes the sequence again. Under the one-token-per-step decoding schedule studied here, generating an answer of length L can require L forward passes. The linear projections in these passes account for most of the arithmetic cost.

Activation sparsity reduces linear-layer computation by omitting selected input entries (Liu et al., 2023; Lee et al., 2024; Liu et al., 2025; Chen et al., 2026). Most training-free methods select entries using their magnitude, sometimes weighted by the corresponding weight-column norm. The omitted entries are replaced by zero. This approach does not exploit a basic property of iterative decoding: a large activation may need little computation if its value has barely changed.

We find that temporal changes are much easier to sparsify than the activations themselves. On the measured LLaDA-8B and Dream-7B trajectories, selecting 10% of entries by weight-informed temporal change leaves 3–4% relative linear-output error. The same budget with weight-informed activation magnitude leaves 41–43% error (Figure 1). The changes are also unevenly distributed across tokens. The newly decoded token accounts for 12–23% of the squared change, although it is only one of hundreds of positions.

We introduce **Temporal Delta Sparsity (TDS)**. Each linear operator stores a reference input and its projected output. At the next step, it forms the difference between current input and that reference. It updates only entries with large weight-informed differences, then adds their projected contribution to the stored output. An entry that is skipped keeps its old reference value. Its outstanding change is therefore reconsidered at the next step. This construction builds on delta networks (Neil et al., 2017); our focus is its allocation and implementation for DLM inference.

TDS uses one budget across tokens and channels within each operator and sequence. A threshold controller adjusts the amount of work over denoising steps. We prove a local accumulator invariant and an error-bound result for selecting the largest scores. These results explain the update rule; they

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.

do not guarantee equality to the dense model's predictions. We also study structured versions that refresh whole token rows or groups of channels, making the saved arithmetic usable by GPU kernels.

In short, our contributions can be summarized as follows:

- We measure the concentration of temporal activation changes in two instruction-tuned DLMs and use it to motivate a training-free sparsity method.
- We develop reference-based updates, a shared token-by-channel budget, and analyses of local error and update counts.
- We compare TDS with six activation-sparsity baselines on four benchmarks. At 88–89% realised sparsity, its average score is close to dense decoding. A row-based implementation also provides measured end-to-end speedups.

2 Related Work

2.1 Diffusion Language Model

Discrete diffusion generates categorical data by reversing a corruption process (Austin et al., 2021a). Masked diffusion uses a special mask token and learns to recover the original tokens (Sahoo et al., 2024; Shi et al., 2024). LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) extend this approach to large language models with instruction-following capabilities. Block diffusion could combine the autoregressive generation across blocks with denoising within each block (Arriola et al., 2025). These models motivate our setting: repeated denoising passes over a sequence create opportunities to reuse computation. TDS accelerates the linear layers without retraining the model or changing its decoding rule.

2.2 Efficient DLM Inference

Bidirectional attention makes the usual autoregressive KV cache inexact, since the existing token representations can change at each step. dLLM-Cache (Liu et al., 2026) selectively refreshes prompt and response features. dKV-Cache (Ma et al., 2025) uses delayed updates to key and value states. Fast-dLLM (Wu et al., 2026) combines the approximate blockwise caching with confidence-based parallel decoding. These methods reduce repeated computation at the token or block level. TDS instead selects updates within linear layers, allowing different channels of a token to be refreshed at different times.

Temporal updates have a longer history. Delta networks (Neil et al., 2017) update stored linear outputs when activations change sufficiently. Moreover, DeltaCNN (Parger et al., 2022) and Eventful Transformers (Dutson et al., 2023) exploit related ideas for video. Chipmunk (Silveria et al., 2025) uses column-sparse deltas to accelerate image and video diffusion transformers. TDS builds on this incremental-computation principle. Its focus is masked language diffusion, with weight-informed selection across token–channel entries and the adaptive allocation of a computation budget over denoising steps.

2.3 Activation Sparsity in LLMs

Activation sparsity reduces linear-layer cost by omitting contributions from selected input entries. Deja Vu (Liu et al., 2023) learns predictors of contextual sparsity. And CATS (Lee et al., 2024) thresholds gate activations, while CHESS (He et al., 2024) uses channel-specific thresholds and selectively sparsifies attention projections. COUNTDOWN (Cheon & Kang, 2025) selects feed-forward channels using up-projection activations. TEAL (Liu et al., 2025) thresholds linear-layer inputs by magnitude; WINA (Chen et al., 2026) also accounts for weight-column norms. BAP (Kim et al., 2026) periodically selects neurons using activation statistics shared across a batch. In addition, Szatkowski et al. (2026) study activation sparsity across model families, including LLaDA, and show that its active-neuron masks change during denoising. These methods approximate activations by setting selected entries to zero. TDS applies sparsity to the difference from a stored reference. This preserves previously computed contributions and uses the budget to update the entries that have changed most.

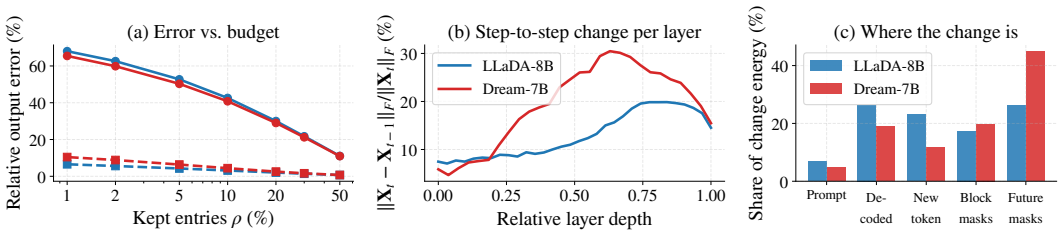

Figure 1: **Activation changes are easier to sparsify than activations.** (a) Relative output error after retaining a fraction ρ of entries by $c_j|Z_{ij}|$. Solid curves use $\mathbf{Z} = \mathbf{X}_t$; dashed curves use $\mathbf{Z} = \mathbf{X}_t - \mathbf{X}_{t-1}$. Results average over operators, layers and steps for eight GSM8K prompts. (b) Relative change $\|\mathbf{X}_t - \mathbf{X}_{t-1}\|_F / \|\mathbf{X}_t\|_F$ by layer, averaged over operators. (c) Fraction of squared activation change contributed by each token type.

3 Background

Notation. Bold uppercase letters denote matrices; X_{ij}, $\mathbf{X}_i$ and $\mathbf{X}_{:,j}$ denote an entry, row and column. Indices t, i and j denote a denoising step, token and input channel. We omit layer and operator indices unless needed. The symbols $\odot$, $\mathbf{1}$ and $\mathbb{1}[\cdot]$ denote element-wise multiplication, an all-ones matrix and an indicator. The quantity $\|\mathbf{M}\|_0$ counts nonzero entries.

3.1 Masked diffusion decoding

A masked DLM starts with a prompt of P tokens and L masked answer positions, for a total length $n = P + L$. At step t, a bidirectional Transformer reads the current sequence $x^{(t-1)}$ and predicts a distribution $p_\theta(\cdot \mid x^{(t-1)})_i$ at each position. Our greedy confidence decoder selects the most confident masked position in the current decoding block:

$$i_t = \arg\max_{i \in \mathcal{I}_t} \max_{v \in \mathcal{V}} p_\theta(v \mid x^{(t-1)})_i, \qquad x_{i_t}^{(t)} = \arg\max_{v \in \mathcal{V}} p_\theta(v \mid x^{(t-1)})_{i_t}, \tag{1}$$

where $\mathcal{I}_t$ is the set of eligible masked positions and $\mathcal{V}$ is the vocabulary. Other positions remain unchanged. This schedule commits one token per step and takes $T \leq L$ full forward passes, with early stopping at an end-of-sequence token. DLMs can also commit several tokens per step; our main experiments use one.

3.2 Linear operators and activation sparsity

Each layer has four linear operators: fused query/key/value, attention output, fused gate/up, and down projections. We write an operator as $\mathbf{Y}_t = \mathbf{X}_t \mathbf{W}^\top$, with input $\mathbf{X}_t \in \mathbb{R}^{n \times d_{\text{in}}}$ and weight $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. The fused projections read normalised activations. The down projection reads $\text{SiLU}(\mathbf{G}) \odot \mathbf{U}$, where $\mathbf{G}$ and $\mathbf{U}$ are the gate and up outputs. These operators account for about 98% of the multiply–accumulates (MACs) at our sequence lengths, $n \approx 350$–440.

Activation sparsity replaces the input with $\mathbf{X}_t \odot \mathbf{M}_t$, where $\mathbf{M}_t$ is a binary mask. Each omitted entry saves d_{out} MACs in the linear product. Magnitude-based methods keep large $|X_{t,ij}|$ (Liu et al., 2025) or weight-informed scores $c_j|X_{t,ij}|$, where $c_j = \|\mathbf{W}_{:,j}\|_2$ (Chen et al., 2026). They approximate omitted entries by zero. We distinguish the target *density* $\rho \in (0, 1]$ from realised *sparsity*, the fraction of linear-layer MACs skipped:

$$\text{Sp} = 1 - \frac{\sum_{t=1}^{T} \sum_o C_t^o + C_{\text{sel}}}{\sum_{t=1}^{T} \sum_o n d_{\text{in}}^o d_{\text{out}}^o}. \tag{2}$$

Here o ranges over the operators and layers, and C_t^o counts the required projection MACs. For an input-entry mask, $C_t^o = \|\mathbf{M}_t^o\|_0 d_{\text{out}}^o$; a dense projection costs $n d_{\text{in}}^o d_{\text{out}}^o$. And the term C_{sel} counts additional selection or proxy MACs without double-counting projections. Section C gives the accounting details.

3.3 Temporal changes are small and concentrated

Changing one input token leaves much of the next step's computation similar. On LLaDA-8B and Dream-7B, the layer-averaged relative activation change is 0.13–0.19 (Figure 1b). At density $\rho = 0.1$, sparsifying this change gives only 3–4% relative output error; sparsifying the activation gives 41–43% (Figure 1a). The newly decoded token alone contributes 12–23% of the squared change (Figure 1c). These observations motivate two choices: reuse earlier activations as references, and share the update budget across all tokens.

4 Temporal Delta Sparsity

4.1 Update only the changes that matter

TDS stores a reference input $\mathbf{R}$ and its linear output $\mathbf{A} = \mathbf{R}\mathbf{W}^\top$ for each operator. The identity $\mathbf{X}\mathbf{W}^\top = \mathbf{A} + (\mathbf{X} - \mathbf{R})\mathbf{W}^\top$ allows us to reuse $\mathbf{A}$ and compute only a sparse correction. A reference entry records the input at its *last refresh*; it need not come from the immediately preceding step. Thus a small skipped change remains in the next residual.

The first step is dense: $\mathbf{R}_1 = \mathbf{X}_1$ and $\mathbf{A}_1 = \mathbf{X}_1\mathbf{W}^\top$. For $t \geq 2$, let $\mathbf{D}_t = \mathbf{X}_t - \mathbf{R}_{t-1}$ be the pending change. A binary mask $\mathbf{M}_t$ selects entries to refresh:

$$\begin{aligned} \mathbf{R}_t &= \mathbf{R}_{t-1} + \mathbf{D}_t \odot \mathbf{M}_t, \\ \mathbf{A}_t &= \mathbf{A}_{t-1} + (\mathbf{D}_t \odot \mathbf{M}_t)\mathbf{W}^\top, \qquad \hat{\mathbf{Y}}_t = \mathbf{A}_t. \end{aligned} \tag{3}$$

Selected reference entries become their current values; all others stay unchanged. This incremental update follows delta networks (Neil et al., 2017), DeltaCNN (Parger et al., 2022) and Eventful Transformers (Dutson et al., 2023). TDS adds a weight-informed selection rule and an adaptive budget for DLM decoding.

Proposition 1 (Accumulator consistency)**.** *In exact arithmetic, Equation* (3) *maintains* $\mathbf{A}_t = \mathbf{R}_t\mathbf{W}^\top$ *for arbitrary inputs and masks. For the dense product of the* current *input,* $\mathbf{Y}_t := \mathbf{X}_t\mathbf{W}^\top$*, its error is*

$$\hat{\mathbf{Y}}_t - \mathbf{Y}_t = -\big[\mathbf{D}_t \odot (\mathbf{1} - \mathbf{M}_t)\big]\mathbf{W}^\top, \qquad \|\hat{\mathbf{Y}}_{t,i} - \mathbf{Y}_{t,i}\|_2 \leq \sum_{j:M_{t,ij}=0} c_j |D_{t,ij}|. \tag{4}$$

The error comes from the pending residual, rather than a sum of discarded updates. This is a *local* statement: $\mathbf{X}_t$ may already contain errors from earlier sparsified layers. It does not guarantee the agreement with a fully dense model or exclude floating-point drift. Proofs and numerical details are in Sections B and C.

4.2 Share the budget across tokens and channels

We score each pending change by its contribution to the bound in Equation (4):

$$s_{t,ij} = c_j |D_{t,ij}|, \qquad c_j = \|\mathbf{W}_{:,j}\|_2, \qquad e_{t,i} = \sum_j s_{t,ij}^2. \tag{5}$$

The matrix $\mathbf{S}_t = [s_{t,ij}]$ contains the entry scores; $e_{t,i}$ is the energy of token i. Column norms are computed from the weights and require no calibration data. We select across the entire $n \times d_{\text{in}}$ matrix of each operator and sequence. Tokens with larger changes can therefore receive more updates.

Proposition 2 (One-step allocation)**.** *Fix* $\mathbf{X}_t$ *and* $\mathbf{R}_{t-1}$*. For* $K \in \{0, \ldots, nd_{\text{in}}\}$*, retaining the* K *largest scores minimises the following bound among masks with at most* K *selected entries:*

$$B_t(\mathbf{M}) := \sum_{(i,j):M_{ij}=0} s_{t,ij}, \qquad \sum_i \|\hat{\mathbf{Y}}_{t,i}(\mathbf{M}) - \mathbf{Y}_{t,i}\|_2 \leq B_t(\mathbf{M}).$$

This result concerns an upper bound for one operator at one step. It does not establish optimality of the actual output error or the full decoding trajectory. For the same entry budget, the bound is no worse than that of a fixed per-token allocation. Setting the reference to zero recovers ordinary activation sparsity (Remark 1).

Algorithm 1 TDS for one linear operator and one sequence.

Require: Inputs $\mathbf{X}_1, \ldots, \mathbf{X}_T$; weights $\mathbf{W}$; target density ρ.
$c_j \leftarrow \|\mathbf{W}_{:,j}\|_2$; $\mathbf{R} \leftarrow \mathbf{X}_1$; $\mathbf{A} \leftarrow \mathbf{X}_1\mathbf{W}^\top$; $N \leftarrow 0$
Output $\hat{\mathbf{Y}}_1 = \mathbf{A}$ ▷ Dense initialisation
for $t = 2, \ldots, T$ **do**
  $\mathbf{D} \leftarrow \mathbf{X}_t - \mathbf{R}$; $s_{ij} \leftarrow c_j|D_{ij}|$
  if $t = 2$ **then**
    $\tau \leftarrow (1-\rho)$-quantile of sampled scores
  $M_{ij} \leftarrow \mathbb{1}[s_{ij} > \tau]$; $K \leftarrow \|\mathbf{M}\|_0$; $N \leftarrow N + K$
  $\mathbf{A} \leftarrow \mathbf{A} + (\mathbf{D} \odot \mathbf{M})\mathbf{W}^\top$; $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{D} \odot \mathbf{M}$
  Output $\hat{\mathbf{Y}}_t = \mathbf{A}$
  $\hat{\rho} \leftarrow K/(nd_{\text{in}})$; $\bar{\rho} \leftarrow N/((t-1)nd_{\text{in}})$
  Update τ using Equation (6)

4.3 Adapt the budget over time

We use a separate threshold for each operator and sequence: $M_{t,ij} = \mathbb{1}[s_{t,ij} > \tau_t]$. This selects the largest scores for its realised count $K_t = \|\mathbf{M}_t\|_0$, so Proposition 2 applies. Let $\hat{\rho}_t = K_t/(nd_{\text{in}})$ be the step's density and $\bar{\rho}_t = (t-1)^{-1}\sum_{u=2}^{t}\hat{\rho}_u$ its mean over sparse steps. The controller adjusts the threshold toward target density ρ:

$$\tau_{t+1} = \tau_t \,\text{clip}\left(\frac{\hat{\rho}_t}{\rho}\right)^{\gamma\alpha} \text{clip}\left(\frac{\bar{\rho}_t}{\rho}\right)^{\gamma(1-\alpha)}, \qquad \text{clip}(r) = \min\{\max\{r, 1/4\}, 4\}. \tag{6}$$

Each factor raises the threshold when its density exceeds the target and lowers it when its density is below the target. We initialise τ_2 from the $(1-\rho)$-quantile of 131,072 sampled scores. We fix $\alpha = 0.4$ and $\gamma = 0.7$ in all experiments; only ρ varies. The controller allows more updates on steps with larger residuals but does not enforce an exact compute budget. Observed mean density exceeds the target by 2–24% in relative terms. We therefore report realised sparsity, including the dense first step. For a *fixed positive threshold*, the number of refreshes is bounded by total activation variation (Proposition 3); this is separate from the adaptive controller.

4.4 Practical variants

TDS applies to all four linear operators in every layer. Fused projections share a mask and use column norms of the stacked weights. References use BF16 and accumulators use FP32; their memory grows linearly with sequence length. Attention, normalisation, RoPE, SiLU and the LM head remain dense.

Unstructured masks give the fine control but are difficult to accelerate with the multi-token matrix multiplication. The *row variant* selects $\text{round}(\rho n)$ tokens with the largest $e_{t,i}$ and recomputes their complete outputs with a dense GEMM. Refreshing a complete row is algebraically equivalent to its delta update. The *chunk variant* sorts tokens by energy, groups them into chunks and selects shared input channels within each chunk. This follows the structured delta-computation of Chipmunk (Sil-veria et al., 2025). Triton kernels (Tillet et al., 2019) implement the chunk updates with split-K scheduling. Section D details both implementations and their differences from Algorithm 1.

A concrete example. Consider one input entry with reference $R = 1$, column norm $c = 1$ and fixed threshold $\tau = 0.1$. If its next values are 1.03, 1.07 and 1.12, the pending residuals are 0.03, 0.07 and 0.12. The first two steps skip this entry. The third refreshes the reference to 1.12 and adds $0.12\mathbf{W}_{:,j}^\top$ to the affected output row. If the input instead returns to its reference, its pending residual becomes zero.

State and work. For the unstructured method, the BF16 reference and FP32 accumulator require $2nd_{\text{in}}+4nd_{\text{out}}$ bytes per operator, excluding weights, column norms and temporary buffers. Scoring examines all nd_{in} input entries, while the selected correction uses $K_t d_{\text{out}}$ MACs. Therefore, sparse multiplication reduces projection work but still incurs selection and memory-access costs.

5 EXPERIMENTS

5.1 EVALUATION PROTOCOL

Models, tasks and decoding. We evaluate LLaDA-8B-Instruct (Nie et al., 2025) and Dream-v0-Instruct-7B (Ye et al., 2025). We report zero-shot accuracy on GSM8K (Cobbe et al., 2021) and MATH500 (Lightman et al., 2024), and pass@1 on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b). Greedy confidence decoding generates up to 256 tokens in blocks of 32, committing one token per step. Development experiments use a separate subset of 256 GSM8K training problems. All experiments use one RTX 5090 GPU; Section C gives implementation and evaluation details.

Baselines and comparison rules. We compare with training-free activation-sparsity methods: TEAL (Liu et al., 2025), WINA (Chen et al., 2026), CHESS (He et al., 2024), COUNTDOWN (Cheon & Kang, 2025) and BAP (Kim et al., 2026). We also apply the top-p rule of Szatkowski et al. (2026) to the FFN intermediate activation. All methods operate on the prompt, decoded and masked positions. Threshold baselines are calibrated on dense trajectories; TDS needs no calibration. By default, TDS uses the entry-level global selection with integral budget control. Sp. reports the realised linear-layer MAC savings, including dense steps and selection projections (Section 3); runtime is measured separately.

Table 1: **Full-test scores (%).** GSM8K and MATH500: accuracy; HumanEval (HEval) and MBPP: pass@1. Avg. and realised sparsity (Sp.) are averaged over tasks. TDS labels give target density ρ; TEAL/WINA labels give target sparsity. Best non-dense scores are bold.

	LLaDA-8B-Instruct						Dream-v0-Instruct-7B					
Method	Sp.	GSM8K	MATH	HEval	MBPP	Avg.	Sp.	GSM8K	MATH	HEval	MBPP	Avg.
Dense	0.0	77.7	31.6	39.0	41.1	47.4	0.0	83.5	42.2	58.5	58.3	60.6
TEAL (50%)	50.0	76.6	33.8	39.0	36.7	46.5	50.2	75.8	33.0	48.8	52.1	52.4
WINA (50%)	50.0	75.0	33.4	36.0	35.5	45.0	50.0	75.1	30.8	43.9	48.3	49.5
CHESS	32.0	54.0	21.4	17.7	16.0	27.3	40.4	36.3	14.0	22.6	28.5	25.3
COUNTDOWN	31.8	70.8	29.8	32.3	36.1	42.3	40.0	70.3	26.8	42.7	44.3	46.0
BAP	30.3	60.2	25.8	23.2	22.6	33.0	38.0	57.3	19.0	28.7	31.5	34.1
Top-p	16.7	75.4	33.6	38.4	37.1	46.1	21.1	77.5	33.6	54.9	53.3	54.8
TDS ($\rho = 0.30$)	68.7	77.6	34.2	36.6	40.9	47.3	68.9	**82.6**	39.6	58.5	**58.3**	**59.8**
TDS ($\rho = 0.10$)	88.6	**79.1**	**35.8**	35.4	**41.1**	**47.8**	88.4	82.0	**41.0**	**62.2**	53.9	**59.8**
TDS ($\rho = 0.05$)	93.9	77.3	34.4	37.2	38.5	46.8	93.6	81.5	34.8	54.3	48.7	54.8
TDS ($\rho = 0.03$)	96.1	77.9	32.4	**40.2**	36.7	46.8	95.7	67.6	20.4	43.9	47.3	44.8
TDS-row ($\rho = 0.10$)	89.0	76.3	32.8	37.2	37.1	45.9	89.1	**82.6**	37.0	54.9	50.7	56.3
TDS-row ($\rho = 0.05$)	93.9	74.2	29.2	29.9	33.7	41.7	94.0	73.7	28.2	45.1	47.9	48.7

5.2 FULL-TEST ACCURACY

Temporal selection preserves the mean score at high sparsity. At $\rho = 0.1$, TDS skips 88.6% of linear-layer MACs on LLaDA and 88.4% on Dream (Table 1). Its mean scores are 47.8% and 59.8%, compared with 47.4% and 60.6% for dense inference. The strongest baseline means are 46.5% for LLaDA (TEAL, 50.0% sparsity) and 54.8% for Dream (top-p, 21.1% sparsity). TDS thus saves more computation at a higher mean score, although LLaDA HumanEval and Dream MBPP fall below their dense scores.

More aggressive sparsity reveals model-dependent limits. At about 94% sparsity, the LLaDA mean remains 46.8%, whereas Dream falls to 54.8%. At about 96%, LLaDA still scores 46.8%, but Dream drops to 44.8%. This difference is consistent with Dream's larger temporal activation changes (Figure 1). Whole-row refresh enables fast dense matrix products (Section 5.6), but at $\rho = 0.1$ its mean is 1.9 points below entry-level selection on the LLaDA and 3.5 points below it on the Dream.

5.3 ACCURACY–SPARSITY FRONTIER ON DEVELOPMENT DATA

Table 2: **Critical sparsity on 256 GSM8K development problems.** Highest tested realised sparsity (%) retaining 99% or 95% of dense accuracy. $<v$: the criterion fails even at the lowest tested sparsity v.

	LLaDA-8B		Dream-7B	
Method	99%	95%	99%	95%
TEAL	<30.7	60.1	<30.3	40.4
WINA	<30.0	50.0	<30.0	30.0
CHESS	<23.2	23.2	<40.3	<40.3
COUNTDOWN	23.9	23.9	<40.2	<40.2
BAP	<30.3	<30.3	<38.0	<38.0
Top-p	20.0	37.6	<26.0	26.0
TDS (ours)	97.8	97.8	68.9	88.5

We adapt critical sparsity (Szatkowski et al., 2026) to count linear-layer MACs rather than zeroed activations (Table 2). At 99% accuracy retention, TDS reaches 97.8% sparsity on LLaDA and 68.9% on Dream; no baseline point above 24% qualifies. At our relaxed 95% criterion, TDS reaches 97.8% and 88.5%.

The 2–2.5-point accuracy standard error makes the 99% criterion noisy. The grid also limits these estimates: several entries are the lowest tested points, and Dream CHESS, COUNTDOWN and BAP each have only one.

5.4 DESIGN ABLATIONS

Table 3: **LLaDA-8B ablations on GSM8K development data.** Accuracy and realised sparsity (Sp.) are percentages; dense accuracy is 81.6%. Compare part (b) with the fixed-budget row in (a); part (c) uses integral control.

	$\rho = 0.10$		$\rho = 0.05$	
Variant	Sp.	Acc.	Sp.	Acc.
(a) Budget control				
TDS (threshold + integral control, default)	88.7	84.4	93.9	84.0
fixed per-step budget (per-step quantile threshold)	89.6	84.8	94.6	82.4
one RMS-scaled threshold for all layers	89.6	80.1	94.6	77.3
(b) Reference, score and granularity (fixed per-step budget)				
zero reference $\mathbf{R} = \mathbf{0}$ (magnitude sparsity)	89.4	0.0	94.1	0.0
unweighted score $\lvert D_{t,ij} \rvert$	89.5	83.6	94.5	80.9
per-token budget	89.6	53.1	94.6	8.6
token-level refresh (whole rows, global budget)	89.0	84.4	93.9	82.0
(c) Hardware-friendly structure (with control)				
chunks of 64 consecutive tokens	87.9	77.7	93.4	28.5
chunks of 32 sorted tokens	88.4	82.4	93.7	81.6
chunks of 64 sorted tokens	88.2	80.9	93.4	76.6

Reference and allocation. With a fixed budget, a zero reference gives 0.0% accuracy at both budgets in Table 3. A per-token budget scores 53.1% and 8.6%, versus 84.8% and 82.4% for global temporal selection. Weighting adds 1.2 and 1.5 points over unweighted scores.

Budget control. Control spends more on difficult steps and helps at extreme sparsity: 82.4% accuracy at 97.8% sparsity, versus 77.7% at 97.6% for a fixed budget (Table 8). These operating points are close, but not identical. A shared RMS-scaled threshold performs worse.

Mask structure. Rows remain competitive at the two budgets in Table 3, but degrade earlier at higher sparsity (Table 8). Consecutive chunks of 64 tokens lose accuracy; sorting by residual energy helps. Sorted chunks of 32 stay within 2.4 points of unstructured selection (Table 9).

Transfer to Dream. Zero-reference selection, per-token budgets, and a shared threshold score below 4% on Dream (Table 7). Control raises accuracy from 74.6% to 81.6% at $\rho = 0.05$; chunking hurts more than on LLaDA.

Table 4: **LLaDA-8B kernel speedup over dense.** Mean of 32 steps on synthetic length-n sequences, batch size 1, one RTX 5090. All rows use CUDA graphs; all chunks use $C = 64$.

	$n = 512$			$n = 1024$		
Implementation	$\rho = 0.2$	0.1	0.05	$\rho = 0.2$	0.1	0.05
consecutive chunks ($C = 64$)	1.29	1.65	2.02	1.36	1.69	2.03
sorted chunks, no split-K	1.22	1.44	1.76	1.37	1.59	1.82
sorted chunks, split-K	1.46	1.74	1.99	1.54	1.88	2.19
token rows	1.65	1.98	2.14	2.01	2.23	2.56

Kernel structure. Sorting improves accuracy but adds overhead: at $n = 512$, $\rho = 0.1$, sorted chunks reach 1.44× without split-K, versus 1.65× for consecutive chunks (Table 4). And split-K balances chunk workloads, raising speedup to 1.74×. Rows are fastest in every setting, using one cuBLAS GEMM per operator. All variants use CUDA graphs, whose effect is not isolated here.

5.5 Analysis of temporal change

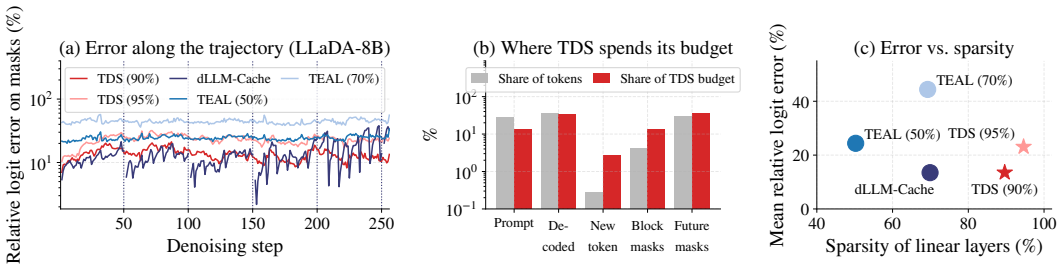

Figure 2: **LLaDA-8B diagnostics with fixed-budget TDS.** (a) Relative logit error on current-block masks, on eight dense teacher-forced development trajectories; dotted lines mark cache refreshes. (b) Position and budget shares by token type, on eight test prompts. (c) Mean error from (a) versus realised sparsity.

Observed error stability. On the same dense teacher-forced inputs, TDS's error remains roughly stable over 256 steps (Figure 2a). Token caching shows a sawtooth pattern between refreshes; TEAL has a larger persistent error. At 90% sparsity, TDS has a mean logit error similar to dLLM-Cache ($I_r = 8$) at about 70% sparsity; at 95%, it is similar to TEAL at 50% (Figure 2c). The committed token agrees with dense inference in 99.5% of steps at 90% sparsity. This small teacher-forced sample does not establish free-running stability; Proposition 1 is a local operator statement.

Where the budget goes. At $\rho = 0.1$, the newly decoded token is about 0.3% of positions but receives roughly ten times that budget share; prompt tokens receive half their proportional share (Figure 2b). This flexibility explains the gain over per-token budgets.

Interpreting budget control. At $\rho = 0.05$, the controller retains 6.1% of LLaDA's linear-layer MACs, versus 5.4% for a fixed budget, while accuracy rises from 82.4% to 84.0% (Table 3). This gain combines adaptive allocation with additional computation. Equal target densities therefore do not isolate the controller's benefit at matched realised sparsity.

5.6 Measured runtime and practical limits

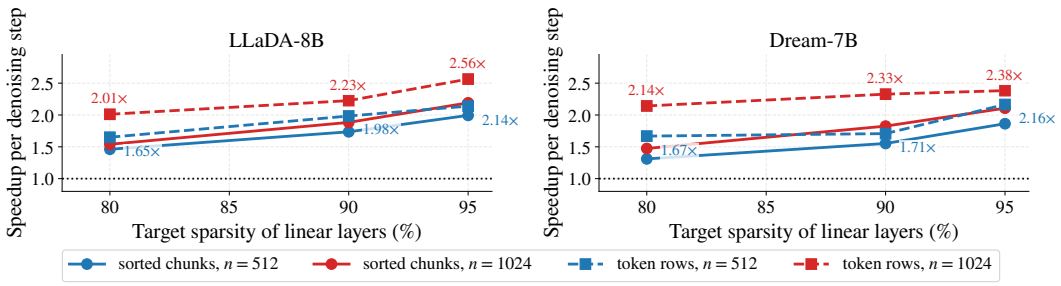

Figure 3: **Measured speedup per step versus target sparsity $1 - \rho$.** Rows use dense GEMMs; sorted chunks use split-K. Each point times 32 steps on synthetic length-n sequences at batch size 1 on one RTX 5090. Both dense and sparse runs use CUDA graphs.

Table 5: **Generation on the first 200 GSM8K test problems.** Batch size 1, one RTX 5090, CUDA graphs for all rows. Sp. and accuracy are percentages. ms/step includes attention, the language-model head and decoding. Speedup uses total generation time, accounting for the different EOS stopping times.

Model	Method	ρ	Sp.	Acc.	ms/step	Speedup
LLaDA-8B	Dense	1.0	0.0	79.0	59.4	1.00×
	TDS-row	0.1	89.6	79.0	31.3	1.90×
	TDS-row	0.05	94.6	76.0	29.8	1.99×
	TDS-chunk	0.2	78.7	80.0	42.1	1.41×
	TDS-chunk	0.1	88.4	81.0	35.8	1.66×
	TDS-chunk	0.05	93.5	73.5	31.7	1.87×
Dream-7B	Dense	1.0	0.0	83.0	45.5	1.00×
	TDS-row	0.1	89.4	85.5	32.2	1.46×
	TDS-chunk	0.1	88.1	78.0	32.7	1.41×
	TDS-chunk	0.05	93.1	27.0	28.8	1.47×

Per-step speed. At $\rho = 0.1$, the row variant speeds up each LLaDA step by $1.98\times$ at $n = 512$ and $2.23\times$ at $n = 1024$ (Figure 3). It reaches $2.56\times$ at $\rho = 0.05$, $n = 1024$. These timings apply only to the structured implementations.

End-to-end speed. On 200 GSM8K test problems, row refresh at $\rho = 0.1$ gives $1.90\times$ speedup on LLaDA at equal accuracy (Table 5). On Dream it gives $1.46\times$, scoring 85.5% versus 83.0%; this subset does not establish an accuracy improvement. Lower budgets trade accuracy for speed: Dream chunks fall to 27.0% at $\rho = 0.05$.

Scope and limitations. Runtime covers one GPU and batch size 1; state memory grows with sequence length (Section D). Parallel decoding can reduce temporal sparsity: committing two tokens per step on LLaDA gives 79.3% accuracy at 89% sparsity with a fixed budget, versus 84.0% for dense.

6 Conclusion

Temporal Delta Sparsity saves computation by updating changes relative to a stored reference. Skipped changes remain available for later steps, while a global budget prioritises changing tokens and channels. Across two models and four benchmarks, entry-level selection skips about 89% of linear-layer MACs within one point of the dense mean score. The row variant yields 1.46–$1.90\times$ speedups on the GSM8K subset. These results support the applications to iterative mathematical reasoning and code generation with masked diffusion models. The trade-offs are additional state and approximation error; larger batches and more parallel decoding remain future work.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing. We have not used generative AI tools for writing code and conceptualizing the method. Additionally, we used generative AI tools to survey the literature in this field. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

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.

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

Sihan Chen, Dan Zhao, Jongwoo Ko, Colby Banbury, Huiping Zhuang, Luming Liang, Pashmina Cameron, and Tianyi Chen. Wina: Weight informed neuron activation for accelerating large language model inference. In *International Conference on Learning Representations*, volume 2026, pp. 122858–122880, 2026.

Jaewon Cheon and Pilsung Kang. Countdown: Contextually sparse activation filtering out unnecessary weights in down projection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 28381–28397, 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.

Matthew Dutson, Yin Li, and Mohit Gupta. Eventful transformers: Leveraging temporal redundancy in vision transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 16865–16877. IEEE, 2023.

Junhui He, Shangyu Wu, Weidong Wen, Chun Jason Xue, and Qingan Li. Chess: Optimizing llm inference via channel-wise thresholding and selective sparsification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18658–18668, 2024.

Yongmin Kim, Shota Takashiro, Yusuke Iwasawa, Takeshi Kojima, and Yutaka Matsuo. Batch-wise adaptive pruning: Periodic neuron activation-aware weight pruning for language reasoning model. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=pWa82BkTPc.

Donghyun Lee, Jaeyong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. CATS: Context-aware thresholding for sparsity in large language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=v3w2a7EInO.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

James Liu, Pragaash Ponnusamy, Tianle Cai, Yoon Kim, Ben Athiwaratkun, et al. Training-free activation sparsity in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 98302–98322, 2025.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dLLM-cache: Accelerating diffusion large language models with adaptive caching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=guLrgDb2jB.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 149009–149033, 2025. doi: 10.52202/085713-4984.

Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. In *International conference on machine learning*, pp. 2584–2593. PMLR, 2017.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan LI. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50608–50646, 2025. doi: 10.52202/085713-1689.

Mathias Parger, Chengcheng Tang, Christopher D Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. Deltacnn: End-to-end cnn inference of sparse frame differences in videos. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12487–12496. IEEE, 2022.

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.

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.

Austin Silveria, Soham V Govande, and Daniel Y Fu. Chipmunk: Training-free acceleration of diffusion transformers with dynamic column-sparse deltas. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

Filip Szatkowski, Patryk Będkowski, Alessio Devoto, Jan Dubiński, Pasquale Minervini, Mikolaj Piórczynski, Simone Scardapane, and Bartosz Wójcik. Universal properties of activation sparsity in modern large language models. In *International Conference on Learning Representations*, volume 2026, pp. 146247–146269, 2026.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

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. In *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026.

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.

A NOTATION AND SCOPE

Table 6 collects the notation used in method and proofs. We analyse one linear operator and discuss affine biases separately. The input $\mathbf{X}_t$ is the input that this operator actually receives, including any approximation introduced by earlier layers. A local exact product $\mathbf{Y}_t = \mathbf{X}_t\mathbf{W}^\top$ need not equal the corresponding activation on a fully dense decoding trajectory.

Table 6: Notation. Sequence length excludes padding. Density counts retained computation; sparsity counts skipped computation.

Symbol	Meaning
n, L, T	Sequence length, maximum answer length, executed steps
t, i, j	Denoising step, token index, input-channel index
$d_{\text{in}}, d_{\text{out}}$	Input and output dimensions of one operator
$\mathbf{X}_t \in \mathbb{R}^{n\times d_{\text{in}}}, \mathbf{W} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$	Current input and fixed weight matrix
$\mathbf{Y}_t = \mathbf{X}_t\mathbf{W}^\top, \hat{\mathbf{Y}}_t = \mathbf{A}_t$	Local dense product and approximate output
$\mathbf{R}_t \in \mathbb{R}^{n\times d_{\text{in}}}, \mathbf{A}_t \in \mathbb{R}^{n\times d_{\text{out}}}$	Reference and accumulated output after step t
$\mathbf{D}_t = \mathbf{X}_t - \mathbf{R}_{t-1}$	Pending change before selection
$c_j = \|\mathbf{W}_{:,j}\|_2$	Weight-column norm
$s_{t,ij} = c_j\|D_{t,ij}\|, \mathbf{S}_t = [s_{t,ij}]$	Entry score and score matrix
$e_{t,i} = \sum_j s^2_{t,ij}$	Token residual energy
$\mathbf{M}_t \in \{0,1\}^{n\times d_{\text{in}}}, K_t = \|\mathbf{M}_t\|_0$	Selection mask and number of selected entries
$\rho, \hat{\rho}_t, \bar{\rho}_t$	Target, current and mean density
τ_t, α, γ	Threshold, controller weight and gain
C^o_t, C_{sel}	Projection MACs and additional selection/proxy MACs
$C, \mathcal{B}_b, \mathcal{J}_b$	Chunk size, token indices and selected channels
$\sigma_{t,bj} = c_j\|\mathbf{D}_{t,\mathcal{B}_b,j}\|_2$	Score for channel j in chunk b
TV_{ij}	Total variation of entry (i,j) along a trajectory
I_r	Response-cache refresh interval in the diagnostic baseline
Sp.	Realised sparsity, reported as a percentage

B PROOFS AND INTERPRETATION

B.1 WHY SKIPPED CHANGES REMAIN AVAILABLE

Proof of Proposition 1. The initial state satisfies $\mathbf{A}_1 = \mathbf{R}_1\mathbf{W}^\top$. If this holds at step $t-1$, linearity gives

$$\mathbf{A}_t = \mathbf{R}_{t-1}\mathbf{W}^\top + (\mathbf{D}_t \odot \mathbf{M}_t)\mathbf{W}^\top = (\mathbf{R}_{t-1} + \mathbf{D}_t \odot \mathbf{M}_t)\mathbf{W}^\top = \mathbf{R}_t\mathbf{W}^\top.$$

Induction proves the invariant for every step. Also, $\mathbf{X}_t - \mathbf{R}_t = \mathbf{D}_t \odot (\mathbf{1} - \mathbf{M}_t)$, so

$$\hat{\mathbf{Y}}_t - \mathbf{Y}_t = (\mathbf{R}_t - \mathbf{X}_t)\mathbf{W}^\top = -[\mathbf{D}_t \odot (\mathbf{1} - \mathbf{M}_t)]\mathbf{W}^\top.$$

For row i, apply the triangle inequality to $-\sum_{j:M_{t,ij}=0} D_{t,ij}\mathbf{W}^\top_{:,j}$. Each term has norm $c_j|D_{t,ij}|$, which gives Equation (4). □

What the invariant guarantees. An unselected reference entry remains unchanged. The next residual therefore compares the new input with the same stored value and includes its net change since the last refresh. This prevents the bookkeeping from discarding pending changes. It does not bound the propagation of approximation errors via subsequent nonlinear layers or future decoding decisions. References and residuals use BF16, while accumulated outputs use FP32. Rounding in subtraction, matrix products and accumulation can violate the exact-arithmetic invariant. The row variant recomputes selected rows directly and avoids incremental accumulation for those rows.

Affine projections. For a fixed bias $\mathbf{b} \in \mathbb{R}^{d_{\text{out}}}$, initialise $\mathbf{A}_1 = \mathbf{X}_1\mathbf{W}^\top + \mathbf{1}_n\mathbf{b}^\top$ and keep the same delta update. Then $\mathbf{A}_t = \mathbf{R}_t\mathbf{W}^\top + \mathbf{1}_n\mathbf{b}^\top$; the bias cancels in the error identity. It is included once, rather than added at every refresh.

B.2 WHY GLOBAL SELECTION MINIMISES THE BOUND

Proof of Proposition 2. For fixed input and reference, all scores are fixed and non-negative. Write

$$B_t(\mathbf{M}) = \sum_{i,j} s_{t,ij} - \sum_{(i,j):M_{ij}=1} s_{t,ij}.$$

The first term is constant. Minimising B_t is therefore equivalent to retaining the largest possible sum of at most K scores. Any set of K largest scores achieves this; ties do not affect the sum. Summing the row bounds in Equation (4) proves the stated error bound. A per-token allocation, or a selection of complete rows, is another feasible mask if it uses at most K entries. Its bound cannot be smaller than this minimum. □

Limits of the result. The triangle inequality ignores cancellation between weight columns. Thus, the mask that minimises this bound need not minimise the actual output error. The result also fixes the current state; it makes no claim about the best sequence of masks over future steps.

Remark 1 (Zero-reference special case). Resetting $\mathbf{R} = \mathbf{0}$ and $\mathbf{A} = \mathbf{0}$ before each step gives $\hat{\mathbf{Y}} = (\mathbf{X} \odot \mathbf{M})\mathbf{W}^\top$. The score $c_j|X_{ij}|$ is WINA's weight-informed criterion, with a global rather than per-token budget. Using $|X_{ij}|$ and TEAL's calibrated per-operator thresholds instead gives TEAL's mask. Our zero-reference ablation retains the global, weight-informed selection rule and changes only the reference.

B.3 COMPUTE WITH A FIXED THRESHOLD

The following result explains the benefit of temporal reuse when activations change little. It applies to a fixed threshold, not directly to the adaptive controller in Equation (6).

Proposition 3 (Total-variation bound)**.** *Consider the exact-arithmetic updates in Equation* (3)*, with a dense first step and* $M_{t,ij} = \mathbb{1}[c_j|D_{t,ij}| > \tau]$ *for a fixed* $\tau > 0$*. Define* $\mathrm{TV}_{ij} := \sum_{t=2}^{T} |X_{t,ij} - X_{t-1,ij}|$*. Entry* (i, j) *is selected at most* $1 + c_j\mathrm{TV}_{ij}/\tau$ *times, including initialisation. Hence*

$$\sum_{t=1}^{T} \|\mathbf{M}_t\|_0 \le n d_{\text{in}} + \frac{1}{\tau} \sum_{i,j} c_j \mathrm{TV}_{ij}. \tag{7}$$

Proof. Fix an entry and write its refresh steps as $1 = t_0 < t_1 < \cdots < t_m$. Between t_{k-1} and t_k, its reference stays at $X_{t_{k-1},ij}$. The refresh rule and triangle inequality imply

$$\tau < c_j|X_{t_k,ij} - X_{t_{k-1},ij}| \le c_j \sum_{u=t_{k-1}+1}^{t_k} |X_{u,ij} - X_{u-1,ij}|.$$

These intervals do not overlap. Summing gives $m\tau \le c_j\mathrm{TV}_{ij}$, including the case $m = 0$. Add the initial dense computation, then sum over entries to obtain Equation (7). □

A constant, nonzero activation needs no refresh after initialisation. A zero-reference rule with the same threshold selects it at every step whenever $c_j|X_{ij}| > \tau$. More generally, temporal compute depends on total change, whereas zero-reference compute depends on how often magnitudes exceed the threshold. This comparison is explanatory: differently sparsified networks can follow different input trajectories. The proof also extends to thresholds bounded below by a known positive constant, replacing τ by that constant. Our adaptive controller has no prescribed positive lower bound or exact budget guarantee.

C EXPERIMENTAL AND IMPLEMENTATION DETAILS

C.1 MODELS, DECODING AND NUMERICAL PRECISION

We implement LLaDA-8B-Instruct and Dream-v0-Instruct-7B in one PyTorch module. Fusing the query/key/value and gate/up weights exposes the same four linear operators to every method. In our implementation check, argmax predictions match the official code at every position. Weights and activations use BF16; incremental accumulation uses FP32. The maximum answer length is 256. We decode blocks of 32 tokens, committing one token per confidence-decoding step. Dream predicts token i from the hidden state at $i-1$ because it is initialised from an autoregressive model. We apply this shift before selecting the most confident position.

A sequence stops after a completed block contains an end-of-sequence token. Committed tokens are never re-masked, so this stopping rule preserves the final answer truncated at that token. Batches use left padding, and attention masks exclude padding keys. Relative-position RoPE attention makes this padding equivalent in exact arithmetic, but BF16 kernels can change outputs with batch shape.

On the first 200 GSM8K test problems, batched dense decoding scores 77.5% on LLaDA and 85.0% on Dream. At batch size 1, the scores are 79.0% and 83.0%. Changing batch shape changes greedy outputs for 12% and 41% of problems, respectively; correctness changes for 6.5% and 5%. We therefore do not interpret differences of roughly two percentage points on this subset as evidence of improvement.

C.2 COMPUTE ACCOUNTING

We count linear-layer MACs on non-padding tokens over each executed step. Dense initialisation, BAP's dense phases and dLLM-Cache refreshes are included. Selection projections are included: dLLM-Cache pays for its value projection, and gate-based or up-based methods pay for the dense projection used to form their masks. The reported sparsity is measured on evaluation data, rather than inferred from the target parameter. It is an arithmetic count; it does not by itself measure runtime or memory traffic.

Methods may execute different numbers of steps before stopping. Dense Dream averages 255.9 steps on GSM8K, and degraded methods can stop at different times. Thus a method's sparsity describes savings against dense execution of its own steps, not necessarily its total compute reduction relative to the dense baseline run. Counting only FFN MACs does not remove the main gap: CHESS, COUNTDOWN and BAP skip 43–49% of FFN MACs in Table 1, top-p skips 24%, and TDS at $\rho = 0.1$ skips 88%.

C.3 BASELINE ADAPTATIONS

We calibrate threshold methods on dense denoising trajectories from 32 prompts: eight GSM8K training problems, eight MBPP prompt-split examples and 16 Alpaca instructions. Calibration includes prompt, decoded and masked positions at every other step. TEAL and COUNTDOWN use per-layer quantiles at the target sparsity. CHESS uses per-channel FFN thresholds. Moreover, COUNTDOWN keeps the up projection dense, then skips gate rows and down-projection columns whose up-activation magnitude is below threshold. WINA selects the largest $c_j|X_{t,ij}|$ entries within each token and does not use its optional orthogonalising transform.

We apply the top-p rule of Szatkowski et al. (2026) per token to the FFN intermediate activation, following their definition and evaluation harness. Their released LLaDA script instead selects jointly across tokens and channels and leaves the gate-projection input dense. The implementation choices should be considered when comparing results. BAP uses eight initial dense steps, followed by repeated cycles of 18 sparse and two dense steps. Its per-sequence neuron memory selects structured FFN neurons.

D GPU IMPLEMENTATIONS

Unless explicitly labelled otherwise, accuracy experiments always use the unstructured method in Algorithm 1. The row and chunk variants trade the selection flexibility for efficient GPU matrix multiplication. Tables 4 and 5 report their measured runtime.

D.1 ROW VARIANT

At each step, the row variant scores all tokens by $e_{t,i}$ and selects $n_{\text{sel}} = \text{round}(\rho n)$ tokens. A fixed count keeps tensor shapes static. One cuBLAS GEMM recomputes the selected outputs from their current inputs. The implementation scatters these outputs into a BF16 cache and copies the selected inputs into the reference. It needs no FP32 output accumulator because selected rows are recomputed directly. At $\rho = 1$, it reproduces the dense logits in our implementation. The complete denoising step is captured in a CUDA graph.

D.2 CHUNK VARIANT

A chunk contains C tokens that share selected input channels. We sort tokens by residual energy before forming chunks. This groups strongly changing tokens together and avoids spending an entire chunk's budget on a single changing token. For token set $\mathcal{B}_b$ and channel j, define

$$\sigma_{t,bj} = c_j \|\mathbf{D}_{t,\mathcal{B}_b,j}\|_2.$$

If $\mathcal{J}_b$ is the selected channel set, the chunk update is

$$\mathbf{A}_{\mathcal{B}_b,:} \leftarrow \mathbf{A}_{\mathcal{B}_b,:} + \mathbf{D}_{t,\mathcal{B}_b,\mathcal{J}_b} (\mathbf{W}_{:,\mathcal{J}_b})^\top.$$

The reference is refreshed at the same selected entries.

Controller differences. The deployed kernel thresholds squared scores at the $\tau_t^{(2)}$. It applies Equation (6) to this squared threshold, whereas the simulated chunk variants threshold unsquared scores. Thus the deployed controller has effective gain $\gamma/2$ on unsquared scores; the simulations use γ. Let $K_t^{\text{chunk}} = \sum_b |\mathcal{J}_b|$ count selected chunk–channel cells. The density on real tokens is $\hat{\rho}_t = \sum_b |\mathcal{B}_b|\,|\mathcal{J}_b|/(nd_{\text{in}})$, which reduces to $K_t^{\text{chunk}} C/(nd_{\text{in}})$ when every chunk has C tokens. The first sparse kernel step initialises its threshold with exact top-K selection over these cells. It does not use the sampled quantile in the unstructured method. After initialisation, only token energies are sorted.

Kernel stages. The batch-size-1 implementation uses chunks of $C = 64$ and six stages:

(1) Compute BF16 residuals and token energies in a Triton kernel.
(2) Sort tokens by energy.
(3) Compute chunk scores in the sorted token order.
(4) Threshold squared scores and compact selected channels using cumulative sums and the scatter.
(5) Gather selected residuals into contiguous buffers and refresh their reference entries.
(6) Multiply each chunk in channel segments and add the results to the FP32 accumulator.

The final stage uses a split-K Triton GEMM with atomic additions. Its segment length is autotuned over $\{256, 512, 1024\}$ for each operator shape. Split-K distributes heavy chunks across multiple GPU work units. A final conversion writes the accumulator to BF16. The model exposes transposed views of its weights for $\mathbf{W}^\top$. No stage requires host synchronisation, so a complete denoising step can be captured and replayed as a CUDA graph. The dense baseline uses the same graph-capture setup.

E ADDITIONAL RESULTS

E.1 DREAM ABLATIONS

Table 7 repeats the main ablations on Dream. With a fixed budget, zero-reference and per-token selection both fall below 4% accuracy. Temporal referencing with global selection reaches 84.0% and 74.6% at $\rho = 0.10$ and 0.05. A single shared threshold also performs poorly. At $\rho = 0.05$, the controller improves accuracy to 81.6%, with somewhat more computation. Weighting scores has mixed effects, so we retain one scoring rule across both models.

Whole-row refresh is competitive under the fixed budget. Chunks lose more accuracy on Dream than on LLaDA. Sorting helps at $\rho = 0.10$: chunks of 32 tokens reach 82.8%. At $\rho = 0.05$, the same variant reaches only 58.2%. This is consistent with Dream's larger activation changes in Figure 1b and the deployed chunk results in Table 5.

Table 7: **Dream ablations.** GSM8K development accuracy and realised sparsity (both in %); dense accuracy is 87.9%. Part (b) uses the fixed budget in part (a), and part (c) uses the controller.

	$\rho = 0.10$		$\rho = 0.05$	
Variant	Sp.	Acc.	Sp.	Acc.
(a) Budget control				
TDS (threshold + integral control, default)	88.5	86.3	93.7	81.6
fixed per-step budget (per-step quantile threshold)	89.5	84.0	94.4	74.6
one RMS-scaled threshold for all layers	89.4	2.7	94.2	2.0
(b) Reference, score and granularity (fixed per-step budget)				
zero reference $\mathbf{R} = \mathbf{0}$ (magnitude sparsity)	89.6	0.4	94.6	0.0
unweighted score $\lvert D_{t,ij} \rvert$	89.3	87.1	94.4	73.0
per-token budget	89.6	3.9	94.6	1.2
token-level refresh (whole rows, global budget)	89.3	86.3	94.2	80.5
(c) Hardware-friendly structure (with control)				
chunks of 64 consecutive tokens	87.8	31.6	93.4	0.8
chunks of 32 sorted tokens	88.3	82.8	93.4	58.2
chunks of 64 sorted tokens	88.0	76.6	93.2	15.6

E.2 BUDGET RULES AND PARALLEL DECODING

Table 8 extends the LLaDA budget ablation. The controller retains at least the dense accuracy of 81.6% through 97.8% sparsity. The fixed budget does so through 96.6% sparsity, then reaches 77.7% at 97.6% sparsity. Whole-row refresh degrades earlier, near 96% sparsity. These are observed values on the evaluated grid, rather than estimated operating limits. In the two-token-per-step experiment, fixed-budget TDS reaches 79.3% accuracy at 89% sparsity, compared with 84.0% for the dense decoding. The larger gap suggests that committing more tokens per step can reduce the benefit of temporal reuse.

Table 8: **Budget rules across the evaluated range.** LLaDA GSM8K development results; dense accuracy is 81.6%. Each row is one run. Sp. and Acc. are realised sparsity and accuracy in %.

Budget rule	ρ	Sp.	Acc.
integral control (default)	0.3	68.8	82.4
	0.1	88.7	84.4
	0.05	93.9	84.0
	0.03	96.1	85.2
	0.02	97.2	82.4
	0.015	97.8	82.4
	0.01	98.4	72.7
fixed per-step budget	0.2	79.6	82.8
	0.1	89.6	84.8
	0.05	94.6	82.4
	0.03	96.6	82.4
	0.02	97.6	77.7
	0.01	98.6	68.4
	0.005	99.1	45.3
whole token rows (fixed budget)	0.1	89.0	84.4
	0.05	93.9	82.0
	0.03	95.9	75.8
	0.02	96.9	68.8

E.3 Chunk size and token order

Table 9 separates the effects of chunk size, token ordering and budget control. At $\rho = 0.1$, fixed-budget chunks of 64 consecutive tokens reach 53.1% accuracy. Sorting recovers much of this loss: chunks of 32 tokens reach 84.0%, close to the unstructured result of 84.8%. At $\rho = 0.05$, sorted chunks of 64 tokens reach 20.3% with a fixed budget and 76.6% with the controller. The latter uses more computation, so this comparison reflects both adaptive allocation and budget differences. On the first 200 GSM8K test problems, an earlier kernel with unsorted chunks reaches 75.0% at $\rho = 0.1$, compared with 81.0% for sorted chunks in Table 5.

Table 9: **Chunk size, token order and budget control.** LLaDA GSM8K development results; dense accuracy is 81.6%. A chunk of C tokens shares one channel set. "Sorted" orders tokens by residual energy. Sp. and Acc. are percentages.

Order	C	Budget	ρ	Sp.	Acc.
consecutive	16	fixed	0.1	89.8	78.9
consecutive	32	fixed	0.1	89.8	76.2
consecutive	64	fixed	0.1	89.9	53.1
consecutive	64	control	0.1	87.9	77.7
consecutive	64	control	0.05	93.4	28.5
sorted	32	fixed	0.1	89.7	84.0
sorted	32	fixed	0.05	94.6	76.6
sorted	32	control	0.1	88.4	82.4
sorted	32	control	0.05	93.7	81.6
sorted	64	fixed	0.1	89.6	82.8
sorted	64	fixed	0.05	94.6	20.3
sorted	64	control	0.1	88.2	80.9
sorted	64	control	0.05	93.4	76.6