Predict, Reuse, and Repair:

ACCELERATING DYNAMIC SPARSE ATTENTION FOR LONG-CONTEXT LLM DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic sparse attention (DSA) accelerates long-context LLM decoding by attending to only the top-K KV blocks relevant to each query. Despite its memory and computing efficiency, DSA also introduces a serialized selection-to-attention dependency that emerges as a new latency bottleneck. We present PRR, a speculate-reuse-repair runtime that exploits temporal locality in DSA selections to predict likely blocks, speculate the attention over them while selection is in flight, and incrementally repair missed blocks once the true selected set is known. PRR uses a lightweight EMA-based predictor, a profiling-guided speculation budget that keeps speculative work off the critical path, and a FlashAttention-based repair kernel that folds missed blocks into the partial attention state using online-softmax statistics. Across long-context benchmarks and both training-free (Quest and InfLLM-v2) and trainable (SeerAttention-R) DSA methods, PRR reduces per-token decoding latency by up to 40% while preserving downstream task accuracy.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed for long-context workloads such as deep research (Zheng et al., 2025; Li et al., 2025), multi-step reasoning (Wang et al., 2024; Yuan et al., 2025b), and tool-using agents (He et al., 2025; Yang et al., 2025). However, decoding over long contexts remains expensive because the runtime has to access and compute over a growing KV cache for every generated token. Dynamic sparse attention (DSA) offers a promising remedy by selecting, at each decoding step, only the top-K Key-Value (KV) blocks most relevant to the current query (Tang et al., 2024; Xiao et al., 2024a; Zhao et al., 2025; Yuan et al., 2025a; Gao et al., 2026). By adapting the sparsity pattern to each query, DSA preserves the most relevant context while substantially reducing attention computation, making it an attractive approach for efficient LLM decoding.

While DSA reduces dense attention arithmetic, it introduces a new bottleneck on the decoding critical path. As shown in Figure 1, each decoding step (i.e., a transformer layer) consists of *selection*, *attention*, and *FFN stages*. Before attention can run, DSA has to first estimate block importance and identify the top-K KV blocks relevant to the current query (within the Selection phase). This selection may use training-free scoring, as in Quest and InfLLM-v2, or a learned gate, as in SeerAttention-R. Because these block identities are unknown until selection completes, attention operation has to wait until selection finishes. As context length grows, this *selection-to-attention* dependency becomes increasingly costly (§2).

Dynamic, Yet Predictable. Although top-K block selections are computed online in DSA, prior works have observed that such selections often exhibit temporal stability across decoding steps (Lee et al., 2024; Levy, 2026). We find that this property also holds broadly for different DSA methods and various workloads. In our measurements on Qwen3-8B, about 69% of selected blocks are reused across consecutive decoding steps (§2.2). This temporal locality opens an opportunity to *overlap block selection with attention execution*, as illustrated in Figure 1. Rather than waiting for the current top-K selection to be done, the runtime can anticipate likely selected blocks and compute attentions speculatively while block selection is still in flight. This would greatly reduce the decoding latency.

Realizing this opportunity, however, requires addressing two challenges.

• *First, temporal locality alone is not sufficient*. Blindly reusing the previous step's top-K block indices for the current step yields only a modest hit rate. Since each missed block must still be

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.

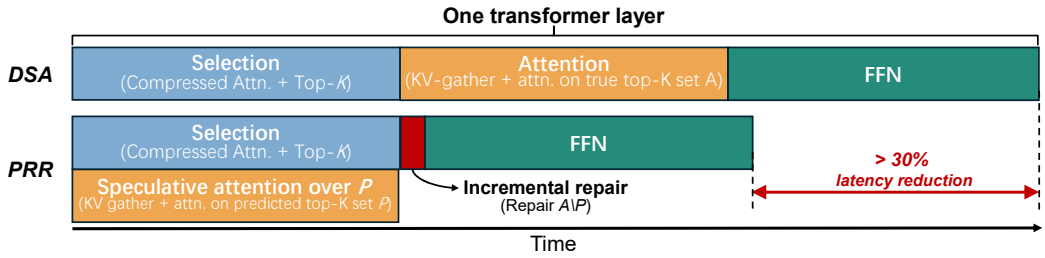

Figure 1: Standard DSA (top) serializes selection, attention over the true top-K block set A, and FFN execution. PRR (bottom) predicts a block set P and executes speculative attention over P in parallel with selection. After the true set A is known, PRR incrementally repairs only the missed blocks $A \setminus P$ before FFN execution, saving over 30% latency for each transformer layer.

incorporated into the final attention output, a low hit rate would significantly increase the incremental repair overhead, limiting the benefit of speculation.

• *Second, speculation must cover the true DSA selections despite inevitable mispredictions*. The final attention output must include all blocks in the true DSA-selected set. However, existing inference engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024), as well as attention kernels such as FlashAttention (Dao, 2024), do not expose an interface for incrementally incorporating missed blocks into a partially computed attention result. As a result, correcting a miss typically requires recomputing attention from scratch over the true selected set, forfeiting much of the latency benefit of speculative attention computation.

In this paper, we present PRR (predict, reuse, and repair), a speculative attention runtime for DSA that addresses both challenges while preserving coverage of the baseline-selected blocks. We first introduce a lightweight, exponential moving average (EMA)-based predictor that tracks the temporal trajectory of per-block importance scores to predict upcoming top-K set P with high accuracy. Its smoothing hyperparameters are calibrated per-prompt using importance scores already produced during prefill, adding negligible model execution cost. To overlap attention computation with block selection, PRR executes speculative attention over this predicted block set P in parallel with the current top-K selection. The size of P is controlled by a profiled dynamic budget that keeps speculative execution off the critical path, i.e., it won't delay FFN. Once the true top-K block set A becomes available, missed blocks in $A \setminus P$ are incorporated through incremental attention repair. To make repair efficient, we further design a customized CUDA kernel based on FlashAttention's online-softmax recurrence that performs *incremental* attention repair. This ensures that all blocks in A are included, while bounding repair work by $|A \setminus P|$ rather than $|A|$.

With the above design, PRR preserves full coverage of the standard DSA selection: although it speculates on likely top-K blocks, the final attention output always covers the true DSA-selected blocks after repairing. Experiments across five representative benchmarks including LongBench (Bai et al., 2024), InfiniteBench (Zhang et al., 2024), RULER (Hsieh et al., 2024), AIME (Hugging Face H4, 2025), and MATH500 (Lightman et al., 2023) show that PRR achieves per-token decoding speedups of up to $1.42\times$, $1.56\times$, and $1.44\times$ over both training-free DSA (Quest, InfLLM-v2), and trainable DSA (SeerAttention-R), respectively, while maintaining the same downstream task accuracy. We also study the latency gain in batched serving setup (§5.6) and ablate the benefit of speculative execution from our kernel optimization (§A.12). Our contributions are listed below:

- We identify the *selection-to-attention* dependency as a new critical-path bottleneck in dynamic sparse attention mechanisms.
- We design PRR, a runtime that overlaps speculative attention over predicted blocks with selection, reuses correctly predicted block states, and repairs only missed blocks.
- We implement an online-softmax-based incremental repair kernel that ensures full top-K coverage as standard DSA and bounds post-selection work by the number of missed blocks.

2 Motivation

In this section, we first show that DSA shifts decoding cost to selection-attention dependency (§2.1). We then show that sparse indices exhibit strong predictability, and that DSA decoding leaves enough idle GPU resources to turn this predictability into an opportunity for speculative attention (§2.2).

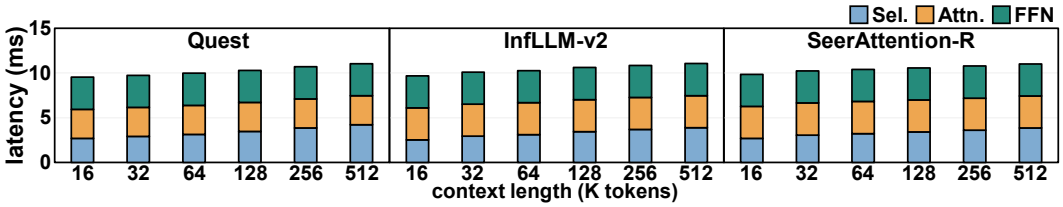

Figure 2: Latency breakdown of Qwen3-8B with Quest, InfLLM-v2, and SeerAttention-R.

Table 1: Overlap rate of blocks in consecutive selections of Qwen3-8B across DSAs and benchmarks.

DSA	Long Bench	Infinite Bench	RULER	AIME	MATH 500	Avg.
Quest	66.7%	68.8%	69.7%	68.9%	70.3%	**68.9%**
InfLLM-v2	68.1%	66.2%	68.2%	71.4%	71.3%	**69.0%**
SeerAttention-R	66.6%	63.8%	65.9%	72.6%	72.1%	**68.2%**

Experiment Setup. We use Qwen3-8B (Team, 2025) to evaluate three representative DSA methods: two training-free DSAs, Quest (Tang et al., 2024) and InfLLM-v2 (Zhao et al., 2025), and one trainable DSA, SeerAttention-R (Gao et al., 2026), across long-sequence benchmarks and a range of context lengths. We run these studies on NVIDIA H100 GPUs, with FlashInfer's `BlockSparseAttention` (Ye et al., 2025) serving as the optimized sparse-attention backend.

2.1 The Selection-to-Attention Dependency

> **Observation One:** DSA significantly reduces attention computation, but exposes a serialized selection-to-attention dependency on the decoding critical path.

Figure 1 (top) shows the computation flow of DSAs. At each layer, the runtime partitions KV tokens into blocks and estimates their importance to the current query using training-free scoring or a learned gate. It then picks the top-K blocks for the current step. Only then can the runtime fetch the selected KV blocks and execute (sparse) attention, followed by the FFN. This creates a strict selection-to-attention dependency. Figure 2 further shows profiled latency of three stages: selection, attention, and FFN. The serialized selection-to-attention path consumes about 62% of decoding latency at 16K tokens and rises to 70% at 512K. Selection alone grows from 2.5 ms to 4 ms over the same range. Since attention is blocked until top-K selection finishes, this growing selection cost directly delays the attention stage and exposes a substantial critical-path bottleneck.

2.2 The Opportunity for Speculation

> **Observation Two:** The top-K blocks in DSAs show strong locality across adjacent decoding steps, which leaves an opportunity for speculative attention.

Prior works (Lee et al., 2024; Takbir et al., 2025) have shown that the same tokens tend to be selected or to receive high attention scores across consecutive decoding steps. Motivated by this, we measure how many blocks are repeatedly selected in consecutive decoding steps on representative long-context benchmarks: LongBench (Bai et al., 2024), InfiniteBench (Zhang et al., 2024), AIME (Hugging Face H4, 2025), MATH500 (Lightman et al., 2023), and RULER (Hsieh et al., 2024).

Table 1 shows that temporal block selection locality is pervasive across different benchmarks and DSA methods. Across LongBench, InfiniteBench, AIME, MATH500, and RULER, the average overlap rates for Qwen3-8B are 68.9%, 69.0%, and 68.2% under Quest, InfLLM-v2, and SeerAttention-R, respectively. We have similar observations on other LLMs (A.3). This result suggests that DSA selections are dynamic but sufficiently predictable for speculation. Instead of waiting for the current top-K selection to complete, the runtime can predict likely selected blocks and begin executing attention over them while selection is still in progress, as shown in Figure 1.

While temporal locality makes speculation possible, it also raises a new question: *does the GPU have enough spare memory-bandwidth and compute resources to execute speculative KV gathering and*

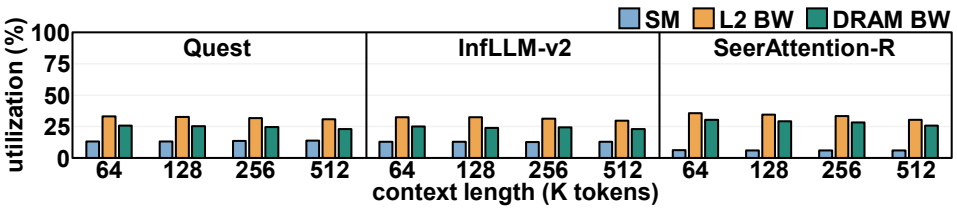

Figure 3: SM utilization, L2 bandwidth, and DRAM bandwidth profiled for Qwen3-8B paired with Quest, InfLLM-v2, and SeerAttention-R across varying context lengths.

attention computation without slowing down the normal DSA pipeline? To answer this question, we profile GPU resource utilization for DSAs.

> **Observation Three:** Long-context DSA decoding leaves sufficient idle GPU resources to support speculative KV gathering and attention computation.

We measure the utilization of streaming multiprocessors (SMs), L2 bandwidth, and DRAM bandwidth during decoding across a range of context lengths, up to 512K tokens. As shown in Figure 3, even at a 512K context length, SM, L2-bandwidth, and DRAM-bandwidth utilization all remain below 40%. This substantial headroom indicates that ample idle resources are available to precompute the sparse attention speculatively, based on estimated block selections, without contending with block selection on the critical path. The same trend holds at larger batch sizes, as shown in A.4.

3 The Design Space

These observations open two interleaved design spaces to explore:

Design Space One: Maximize speculative attention accuracy. Let A be the set of blocks chosen by the ground-truth top-K selection in the current round, and P denote the predicted blocks. We categorize the prediction errors into two types:

- *Missed Blocks ($M = A \setminus P$)*: these blocks lie on the critical path: every block in M should be migrated and attended to after the true selection completes, and the partial attention output have to be corrected accordingly.
- *Wasted Blocks ($N = P \setminus A$)*: these blocks consume extra bandwidth and computing resources. They do not extend the critical path as long as speculative attention fits within the selection window.

The two costs are therefore asymmetric, and the dominant objective is to minimize $|M|$, as shown in Figure 5:

$$\min \ |M| = |A \setminus P| \quad \text{s.t.} \quad |P| \leq \delta |A| \tag{1}$$

where δ is the budget ratio that bounds how many blocks the speculator may fetch. A larger δ admits more candidates and likely reduces $|M|$, but increases bandwidth pressure and speculative attention computation cost. However, if too many blocks are included, speculative attention may take longer than block selection, extending the critical path and negating the latency gains from speculation.

Design Space Two: Enable incremental attention repair. Even with a high-quality predictor, mispredictions are unavoidable, i.e., $|M| > 0$. Once the true top-K set A is known, the runtime must reconcile the speculative attention result with the true selected blocks. A simple fallback is to discard the speculative result and recompute attention over A from scratch. This preserves correctness, but it puts the attention computation back on the critical path with no latency benefit.

4 Design

To explore these design spaces, we propose PRR, a *speculation-and-repair* runtime that predicts likely blocks (§4.1) and overlaps sparse attention with block selection. The speculator is calibrated online with negligible delay (§4.2). PRR then incrementally incorporates missed blocks (§4.3) and adapts the speculative set P to maximize coverage of A without extending the critical path (§4.4).

4.1 The Lightweight, EMA-based Predictor

PRR requires a predictor that is accurate enough to cover the true top-K blocks as much as possible, yet lightweight enough to run off the critical path. We therefore choose a training-free, ultra-

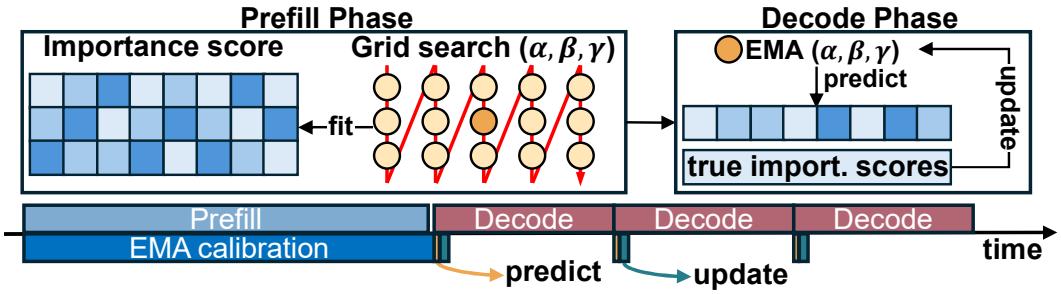

Figure 4: In prefill phase, PRR calibrates EMA over importance scores with grid search. In decode phase, EMA predicts importance scores and updates from true scores produced by block selection.

lightweight exponential moving average (EMA)-based predictor that operates directly on the block-importance scores already produced by DSA.

We denote by IS_i^t the true importance score of block i at decoding step t, produced by the DSA's top-K block selection stage. Before this stage runs at step t, PRR predicts the score of each block as $\widehat{IS}_i^t$ using only historical scores observed up to step $t-1$. The predicted scores are used to select a speculative block set P, with the goal of maximizing coverage of the true top-K A set under a bounded speculation budget.

State and initialization. For each block i, the predictor maintains two state variables after each step t: a smoothed level ℓ_i^t and a trend estimate v_i^t. When block i first enters the KV cache at step τ, we initialize $\ell_i^\tau = IS_i^\tau$ and $v_i^\tau = 0$ because no prior trend is available.

Prediction. At the beginning of step t, before the current DSA selection stage produces IS_i^t, the predictor extrapolates from the previous state:

$$\widehat{IS}_i^t = \ell_i^{t-1} + \gamma v_i^{t-1}, \tag{2}$$

where $\gamma \in [0, 1]$ controls how aggressively the predictor extrapolates the recent trend.

Update. After the DSA selection stage at step t produces the true score IS_i^t, the predictor updates its state using the following equations:

$$\begin{aligned} v_i^t &= \beta\big(IS_i^t - \ell_i^{t-1}\big) + (1-\beta)v_i^{t-1}, \\ \ell_i^t &= \alpha IS_i^t + (1-\alpha)\ell_i^{t-1}. \end{aligned} \tag{3}$$

Here, $\alpha \in (0, 1]$ controls how quickly the level follows newly observed scores, while $\beta \in (0, 1]$ controls how quickly the trend estimate adapts to score changes. Because the predictor consumes only block-importance scores, the same mechanism applies to training-free selectors such as Quest and InfLLM-v2 and to learned selectors such as SeerAttention-R. We discuss the predictor's generality in A.5 and evaluate its hit rate across all three DSAs in §5.4.

4.2 Online Predictor Calibration

The EMA predictor is lightweight, but its prediction accuracy depends on the smoothing hyperparameters (α, β, γ). These parameters determine whether the predictor tracks block-importance dynamics smoothly or reacts quickly to recent changes. A single fixed (α, β, γ) setting would not be effective because the task type and context length can substantially change the stability of the top-K trajectory. PRR instead uses the prefill phase to calibrate these parameters per prompt, as depicted in Figure 4. Because DSA already computes block-importance scores during prefill, prefill provides a ready-made trajectory of block scores at no additional model-execution cost. PRR searches over a small (α, β, γ) candidate grid on this trajectory and selects the setting that best predicts prefill top-K selections, yielding a prompt-adaptive predictor before decoding starts. We elaborate on the design below.

Prefill trajectory. Let T_p denote the number of prefill tokens. The prefill phase produces the matrix $IS \in \mathbb{R}^{T_p \times N_p}$, where row ϕ is the block-importance score vector at prefill token ϕ over the N_p dynamic blocks formed during prefill. This matrix is fully available before the first decoding step.

Calibration objective. Given a candidate hyperparameter setting $\theta = (\alpha, \beta, \gamma)$, we simulate the predictor over IS from prefill, producing a predicted selection set $P_\phi(\theta)$ for each prefill token ϕ, and

measure the resulting score-weighted hit rate:

$$H\big(\theta\,;\,IS\big) \;=\; \frac{1}{T_p}\sum_{\phi}\frac{\sum_{b\in P_\phi(\theta)\cap A_\phi} IS_{\phi,b}}{\sum_{b\in A_\phi} IS_{\phi,b}} \tag{4}$$

where A_ϕ is the ground-truth top-K block selection set at prefill token ϕ and $IS_{\phi,b}$ is the importance score of block b at token ϕ. The per-prompt hyperparameter choice is formulated as:

$$\theta^\star = \arg\max_{\theta} H\big(\theta; IS\big) \tag{5}$$

Search procedure. We perform grid search rather than gradient-based solutions for two reasons. First, gradient-based hyperparameter optimization of a set-valued objective (top-K membership) requires a differentiable surrogate for argsort, which introduces its own approximation error and implementation dependency. Second, the search space is small, so a direct search is efficient.

We set the search grid in our system as follows: $\alpha \in [0.2, 0.8]$ with step 0.2, $\beta \in [0.1, 0.5]$ with step 0.1, and $\gamma \in [0, 0.75]$ with step 0.25, yielding $|\Theta| = 80$ candidates per prompt. The search overlaps with the prefill phase and adds minimal extra latency (0.06 ms) to the critical path (A.6).

4.3 Incremental Repair via Online Softmax

Since mispredictions are unavoidable (i.e., $M = A \setminus P \neq \emptyset$) and softmax is sensitive to the maximum logit, ignoring M produces an inexact output, and the error compounds across decoding steps. We propose a repair phase whose cost scales only with $|M|$, built on FlashAttention's online-softmax recurrence (Dao, 2024). Our customized CUDA kernel enables incremental attention repair.

Instrumented speculative kernel. We modify the FlashAttention forward to write not only the attention output O but also the denominator ℓ and the running maximum m to HBM. The change is surgical: the inner-loop arithmetic is unchanged, and the additional store amounts to two scalars per head, so the kernel retains the memory-access pattern and occupancy of the original forward. The extra bandwidth is negligible relative to KV traffic that already dominates the pass.

Repair kernel. Given $(O^{\text{spec}}, \ell^{\text{spec}}, m^{\text{spec}})$ from the speculative sparse attention and the KV entries of M, a second kernel applies the recurrence in Equations equation 6–equation 8 once per missed block:

$$m^{(t+1)} = \max(m^{(t)}, \tilde{m}) \tag{6}$$

$$\ell^{(t+1)} = e^{m^{(t)} - m^{(t+1)}}\,\ell^{(t)} + e^{\tilde{m} - m^{(t+1)}}\,\tilde{\ell} \tag{7}$$

$$O^{(t+1)} = \frac{e^{m^{(t)} - m^{(t+1)}}\,\ell^{(t)}}{\ell^{(t+1)}}\,O^{(t)} + \frac{e^{\tilde{m} - m^{(t+1)}}}{\ell^{(t+1)}}\sum_{j} e^{S_{t+1,j} - \tilde{m}}\,V_{t+1,j} \tag{8}$$

Each query is handled independently. The missed blocks are streamed through on-chip memory in the same tiled fashion as the FlashAttention forward, so no intermediate score matrix is materialized. The kernel reads O^{spec}, ℓ^{spec}, m^{spec}, and the missed (K, V) tiles once, and writes the corrected (O, ℓ, m) once. Both FLOPs and memory traffic scale solely with $|M|$.

The repaired output covers $P \cup A$: all baseline-selected blocks are included, but overpredicted blocks $P \setminus A$ remain in the softmax. We therefore distinguish numerical equivalence to a single-pass computation over this union from bitwise equivalence to the baseline over A alone. Sections 5.5 and A.11 evaluate downstream accuracy, while A.14 studies numerical equivalence and cross-architecture portability.

4.4 Critical-Path-Aware Speculation Budget

Recall that a larger δ allows PRR to include more predicted blocks in speculative attention, which increases the likelihood that P covers the true selected set A and reduces the missed blocks $A \setminus P$. However, it also increases speculative KV movement and speculative attention computation, which may extend the critical path if speculative attention takes longer than selection.

PRR therefore chooses δ dynamically rather than fixing it globally. For each model-hardware-DSA configuration, we perform a one-time offline profile that measures selection latency and speculative-attention latency under different context lengths and budget ratios. Because the model, hardware

Table 2: Decoding speedup of PRR relative to the serial DSA execution ($1.00\times$). Higher is better.

	Quest						InfLLM-v2					
Model	**Long Bench**	**Infinite Bench**	**RULER**	**AIME**	**MATH 500**	**Avg.**	**Long Bench**	**Infinite Bench**	**RULER**	**AIME**	**MATH 500**	**Avg.**
GLM-4	1.50×	1.40×	1.45×	1.30×	1.42×	**1.42×**	1.64×	1.38×	1.58×	1.62×	1.59×	**1.56×**
GLM-Z1	1.48×	1.39×	1.43×	1.33×	1.45×	**1.42×**	1.61×	1.35×	1.61×	1.64×	1.55×	**1.55×**
DeepSeek-R1	1.39×	1.42×	1.47×	1.35×	1.39×	**1.41×**	1.30×	1.48×	1.61×	1.52×	1.28×	**1.44×**
Llama3-8B	1.36×	1.40×	1.41×	1.39×	1.34×	**1.38×**	1.32×	1.45×	1.55×	1.55×	1.30×	**1.43×**
Qwen3-8B	1.40×	1.37×	1.43×	1.41×	1.40×	**1.41×**	1.48×	1.39×	1.51×	1.52×	1.47×	**1.48×**
Qwen3-14B	1.44×	1.27×	1.41×	1.44×	1.45×	**1.40×**	1.56×	1.43×	1.51×	1.53×	1.53×	**1.51×**
Qwen3-32B	1.26×	1.34×	1.27×	1.27×	1.26×	**1.28×**	1.35×	1.27×	1.29×	1.32×	1.35×	**1.31×**
	SeerAttention-R											
Qwen3-8B	1.44×	1.40×	1.42×	1.48×	1.49×	**1.44×**						
Qwen3-14B	1.42×	1.33×	1.44×	1.43×	1.43×	**1.41×**						
DeepSeek-R1	1.38×	1.41×	1.44×	1.42×	1.43×	**1.42×**						

backend, DSA method, block size are fixed, these latencies are primarily determined by context length and δ. PRR stores the profiling results in a lightweight lookup table indexed by context length, and at runtime selects the largest safe δ whose speculative attention latency fits within the current selection window. This ensures that speculative execution remains aligned with block selection and does not extend the critical path. In our implementation, we profile context lengths of 4K, 8K, 16K, 32K, and 64K, with details and profiling overhead reported in A.7. We use five context lengths as an example, one may profile in a much finer-grained manner to achieve more latency reduction.

5 Evaluation

5.1 Experiment Setups

Datasets. We evaluate PRR on five benchmarks covering long-context understanding (e.g., summarization) and complex reasoning (e,g., math). Long-context tasks are assessed using LongBench (Bai et al., 2024), InfiniteBench (Zhang et al., 2024), and RULER (Hsieh et al., 2024), while reasoning capability is measured on AIME (Hugging Face H4, 2025) and MATH500 (Lightman et al., 2023).

Models. We conduct experiments on seven LLMs: GLM-4-9B-1M (GLM et al., 2024), GLM-Z1-9B (GLM et al., 2024), DeepSeek-R1-8B (DeepSeek-AI, 2025), Llama3-8B-1M (AI@Meta, 2024), Qwen3-8B, Qwen3-14B, and Qwen3-32B (Team, 2025).

DSAs. We evaluate on two training-free DSAs , Quest (Tang et al., 2024) and InfLLM-v2 (Zhao et al., 2025), and one trainable DSA, SeerAttention-R (Gao et al., 2026).

Testbed. We execute experiments on H100 GPUs with a tensor parallelism degree of 2 and BF16 precision. We optimize baselines with Triton and FlashInfer kernel implementations. Full kernel versions, block sizes, top-K and cache budgets are listed in Appendix A.1.

5.2 End-to-end Decoding Latency

We measure end-to-end decoding latency across all models and benchmarks, and report the speedup of PRR over the standard DSA execution in Table 2. As shown, PRR delivers consistent speedups, achieving an average of $1.39\times$, $1.47\times$, and $1.42\times$ speedup under Quest, InfLLM-v2, and SeerAttention-R, respectively. These gains come from three complementary mechanisms. First, the EMA predictor achieves an average top-K overlap rate of roughly 91%, enabling most of the attention work done during speculation. Second, our customized sparse attention CUDA kernel substantially outperforms FlashInfer's `BlockSparseAttention` (§5.4). The headroom it creates lets PRR include more blocks in speculation, lifting the overlap rate from 91% to 98% at no added latency cost. Third, on block misses, incremental attention repair patches in the missing blocks rather than recomputing attention from scratch, preserving the latency savings from speculation.

5.3 Ablation Study

We quantify the contribution of each design component to the overall latency reduction by progressively enabling three features: *Reuse the top-K blocks in the last step:* Speculatively compute

Table 3: Decoding speedup of ablation stages across benchmarks for Qwen3-8B under Quest, InfLLM-v2, and SeerAttention-R, relative to the serial execution ($1.00\times$). Higher is better.

	Quest			InfLLM-v2			SeerAttention-R		
Benchmark	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
LongBench	1.00×	1.19×	1.40×	1.00×	1.25×	1.48×	1.01×	1.23×	1.44×
InfiniteBench	1.00×	1.18×	1.37×	1.01×	1.20×	1.39×	1.00×	1.19×	1.40×
RULER	1.02×	1.20×	1.43×	1.00×	1.27×	1.51×	1.00×	1.22×	1.42×
AIME	1.01×	1.22×	1.41×	1.02×	1.25×	1.52×	1.01×	1.23×	1.48×
MATH500	1.01×	1.18×	1.40×	1.01×	1.24×	1.47×	1.01×	1.25×	1.49×
Avg.	**1.01×**	**1.19×**	**1.41×**	**1.01×**	**1.24×**	**1.48×**	**1.00×**	**1.22×**	**1.44×**

Table 4: Sparse-kernel speedup over FlashInfer's `BlockSparseAttention`.

BS	**1024**	**2048**	**4096**	**6144**	**8192**	**Avg.**
16	1.17×	1.28×	2.00×	2.83×	3.68×	**2.19×**
32	1.17×	1.38×	2.09×	2.78×	3.69×	**2.22×**
64	1.17×	1.38×	2.26×	2.91×	3.64×	**2.27×**
128	1.17×	1.41×	2.23×	3.00×	3.71×	**2.31×**
256	1.03×	1.11×	2.00×	2.75×	3.64×	**2.11×**

Table 5: Overlap (%) between EMA predictions and the true top-K set on Qwen3-8B.

DSA	**Long Bench**	**Infinite Bench**	**RULER**	**AIME**	**MATH 500**	**Avg.**
Quest	98.46	98.04	98.69	97.83	98.24	**98.25**
InfLLM-v2	98.27	97.34	98.42	97.64	98.14	**97.96**
SeerAttention-R	96.16	93.77	95.70	98.77	98.46	**96.57**

attention by using the block selections from the previous decoding step. *Incremental attention repair:* Replace FlashInfer's `BlockSparseAttention` with our customized CUDA kernel, which performs incremental attention repair on missed blocks. *EMA predictor:* Apply our EMA-based predictor to anticipate the upcoming block selections and speculate over the predicted set.

These features are enabled incrementally from S1 to S3, where S3 represents the standard PRR configuration. We use Qwen3-8B as the running example. The consistent trend is observed across other LLMs under all three DSAs, Quest, InfLLM-v2, and SeerAttention-R, listed in A.9.

Results in Table 3 reveal the following. Naively enabling speculative attention through S1 yields negligible improvement over standard DSA execution. Although the block selections reused from the previous decoding step overlap with the current true top-K set by roughly 68% on average, a single missed block triggers a full attention recomputation over the true top-K selections, which offsets the latency savings from speculation. S2 introduces our customized sparse attention kernel, which allows the run-time to repair the missing 32% blocks, raising the average speedup to $1.22\times$. S3 (PRR) attains 98% overlap rate between the predicted top-K set and the true top-K set, leading to $1.44\times$ speedup on average.

5.4 Kernel Speed Comparison

We compare our customized sparse attention kernel against FlashInfer's `BlockSparseAttention` (Ye et al., 2025) on H100 under a GQA configuration with 32 query heads, 8 KV heads, head dimension 128, and a 128K-token context. We sweep the token budget from 1K to 8K and the KV-cache block size from 16 to 256 to reflect realistic sparse attention workloads during decoding.

As shown in Table 4, our kernel outperforms `BlockSparseAttention` across every configuration, with an average speedup of $2.22\times$ and a peak of $3.71\times$ at an 8K budget. At 1K tokens, the speedup is modest ($1.03-1.17\times$) because absolute latencies are small and fixed overheads dominate, whereas at 8K the bandwidth-saturation advantages compound, yielding $3.64-3.71\times$. Across block sizes, performance is stable. Block size 128 yields the highest average speedup ($2.31\times$) and 256 the lowest ($2.11\times$), since the largest tiles approach the dense regime where block-table indirection becomes pure overhead for `BlockSparseAttention`. Beyond raw speedup, our kernel exposes an interface for incremental attention repair that patches missed blocks into an existing attention output rather than recomputing from scratch, which is the key capability underlying PRR's end-to-end gains.

EMA Hit Rate. Table 5 reports the overlap between EMA's predictions and the true top-K set. The predictor achieves 98.25%, 97.96%, 96.57% under Quest, InfLLM-v2, and SeerAttention-R, respectively, across benchmarks. A consistent trend is observed for other LLMs, listed in A.10.

5.5 Downstream Task Accuracy

PRR covers every block selected by the baseline DSA but additionally retains the overpredicted blocks $P \setminus A$, so its output is not bitwise identical to the baseline (§4.3). We therefore verify whether this residual would translate into a quality loss. Table 6 reports the difference in downstream task score between PRR and its baseline for Qwen3-8B under three DSAs: a positive value means PRR improves accuracy. As one can observe, PRR doesn't degrade performance because PRR never drops important blocks, and it occasionally improves slightly when extra included blocks are helpful. A consistent trend is observed for other LLMs in A.11.

Table 6: Min/max/average accuracy difference (PRR-baseline) for Qwen3-8B.

DSA	Min	Max	Avg.
Quest	+0.00	+0.02	+0.00
InfLLM-v2	+0.00	+0.01	+0.00
SeerAttention-R	+0.00	+0.02	+0.00

5.6 Impact of Batch Size

We also measure PRR under various batch sizes. We fix the context length at 16K tokens per prompt and sweep batch sizes of 1, 4, 8, and 16, corresponding to total context lengths of 16K, 64K, 128K, and 256K, respectively. We measure the speedup of PRR relative to three serial DSAs.

Table 7 shows that PRR maintains a speedup of at least 1.245× at larger batch sizes. Together with the utilization measurements in Table 9, these results suggest that resource headroom remains available for speculation–selection overlap even under batched execution. However, as the batch grows, batched sparse attention accounts for a large portion of execution time. Consequently, concurrent execution can hide only a limited part of cost. But this degradation regime is not the common one for long-context serving: the KV cache, rather than compute, caps the achievable batch size. Long-context deployments operate at small batches, which is precisely where PRR's speculation-selection overlap is most effective.

Table 7: Batched decoding speedup of Qwen3-8B across the batch. Higher is better.

DSA	BS=1	BS=4	BS=8	BS=16
Quest	1.413×	1.433×	1.388×	1.335×
InfLLM-v2	1.501×	1.440×	1.325×	1.274×
SeerAttention-R	1.442×	1.313×	1.279×	1.245×

6 Related Work

KV Cache Retrieval for Dynamic Sparse Attention. To avoid the information loss of KV-dropping methods (Zhang et al., 2023; Dong et al., 2024; Xiao et al., 2024b), KV retrieval approaches (Lee et al., 2024; Wu et al., 2025) keep the full KV cache in CPU memory and fetch only query-relevant tokens to reduce transmission, which is orthogonal to PRR. LouisKV (Wu et al., 2025) and AsyncSpade (Luo et al., 2025) predict the query state to prefetch relevant tokens, but approximate query-state estimation can drop important tokens and degrade quality. PRR instead prefetches based on the temporal locality of historical importance scores and speculatively computes attention, with missed tokens reincorporated via attention repair for zero accuracy degradation.

Temporal Locality for KV Cache Management. FlexiCache (Takbir et al., 2025) exploits temporal stability *across heads*: stable heads retain only their top-K KV pages on GPU, while unstable heads keep all pages. This is complementary to PRR, which exploits temporal locality *across decoding steps* — the tendency for the same blocks to be reselected at consecutive steps. The two are composable: PRR's EMA predictor can guide which pages to retain within FlexiCache's stable heads, while PRR's speculative attention separately improves decoding throughput.

7 Conclusion

We have introduced PRR, a novel speculative attention runtime that breaks the selection-to-attention dependency in DSA. To achieve this, we propose a lightweight EMA-based predictor and implement a customized sparse attention CUDA kernel that enables incremental attention repair. Experiments on diverse LLMs and benchmarks show that PRR achieves consistent speedups over serial DSAs.

AI USE STATEMENT

In this work, we used generative AI tools in two ways. First, as a coding assistant to help implement parts of our system; all AI-assisted code was reviewed, tested, and validated by the authors. Second, to rephrase and polish the writing of this paper for clarity and readability. It was not used for research ideation, experimental design, data analysis, or generating any of the results, claims, or findings reported in this paper. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including all text and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. Our contribution is a systems-level inference runtime (PRR) that accelerates the decoding of existing dynamic sparse attention methods. It preserves standard DSA semantics—the final output always covers the true DSA-selected block set after repair—so it does not modify the underlying model's predictions, and therefore introduces no new capabilities, biases, or safety behaviors beyond those already present in the base models we accelerate. Any potential for misuse is inherited from large language models in general and is neither introduced nor amplified by our method.

Our experiments use only publicly available, widely adopted benchmarks (LongBench, InfiniteBench, RULER, AIME, and MATH500) and publicly released pretrained models, all employed in accordance with their licenses and intended research use. The study involves no human subjects, no crowd-sourced annotation, and no collection or processing of personal, private, or sensitive data, so no ethical review board approval was required. We properly credit the datasets, models, and baseline methods (Quest, InfLLM-v2, and SeerAttention-R) on which we build.

We see a modest positive societal impact: by lowering the per-token latency and compute of long-context decoding, PRR can reduce the energy and hardware resources needed to serve large language models. We report all methods and results transparently to support reproducibility, and we are not aware of any conflicts of interest associated with this work.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, February 2025. URL https://matharena.ai/.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, Hayden So, Yu Hua, Ting Cao, Fan Yang, and Mao Yang. Sparse attention adaptation for long reasoning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 111697–111714, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b56d827a2b8433517e722e0272c7f464-Paper-Conference.pdf.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Zhengding Hu, Vibha Murthy, Zaifeng Pan, Wanlu Li, Xiaoyi Fang, Yufei Ding, and Yuke Wang. Hedrarag: Co-optimizing generation and retrieval for heterogeneous rag workflows. In *Proceedings of the ACM SIGOPS 31st symposium on operating systems principles*, pp. 623–638, 2025.

Hugging Face H4. Aime 2024. `https://huggingface.co/datasets/HuggingFaceH4/aime_2024`, 2025. Dataset of 30 problems from the 2024 AIME I and AIME II competitions.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Igorevich Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL `https://openreview.net/forum?id=u7m2CG84BQ`.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 155–172, 2024.

Noam Levy. Dynamic sparse attention: Access patterns and architecture. *arXiv preprint arXiv:2603.13430*, 2026.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context language models understand long contexts? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.859. URL `https://aclanthology.org/2024.acl-long.859/`.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=7LKKHBAMzH`.

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.

Shuqing Luo, Yilin Guan, Pingzhi Li, Hanrui Wang, and Tianlong Chen. Asyncspade: Efficient test-time scaling with asynchronous sparse decoding. *arXiv preprint arXiv:2510.07486*, 2025.

Yanlin Qi, Xinhang Chen, Huiqiang Jiang, Qitong Wang, Botao Peng, and Themis Palpanas. Pariskv: Fast and drift-robust kv-cache retrieval for long-context llms. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.07721.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1348–1362. IEEE, 2025.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024.

Nazmul Takbir, Hamidreza Alikhani, Nikil Dutt, and Sangeetha Abdu Jyothi. Flexicache: Leveraging temporal stability of attention heads for efficient kv cache management, 2025. URL https://arxiv.org/abs/2511.00868.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Team GLM. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024. URL https://arxiv.org/abs/2406.12793.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6106–6131, 2024.

Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian. Using analytical performance/power model and fine-grained dvfs to enhance ai accelerator energy efficiency. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, pp. 1118–1132, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400706981. doi: 10.1145/3669940.3707231. URL https://doi.org/10.1145/3669940.3707231.

Wenbo Wu, Qingyi Si, Xiurui Pan, Ye Wang, and Jie Zhang. Louiskv: Efficient kv cache retrieval for long input-output sequences. *arXiv preprint arXiv:2510.11292*, 2025.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in neural information processing systems*, 37:119638–119661, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024b.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=tXqLxHlb8Z.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL https://arxiv.org/abs/2501.01005.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025a.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Boji Shan, Zeyuan Liu, Jia Deng, Huimin Chen, Ruobing Xie, et al. Advancing llm reasoning generalists with preference trees. In *International Conference on Learning Representations*, volume 2025, pp. 24897–24919, 2025b.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.814. URL https://aclanthology.org/2024.acl-long.814/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, et al. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. *arXiv preprint arXiv:2509.24663*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

A Appendix

A.1 Configurations and Baseline Tuning

To make the speedup numbers in Section 5 auditable, we state the full configuration used throughout. All runs use `bf16` precision and FlashInfer v0.6.3 on H100 GPUs with CUDA 12.8 and a tensor parallelism degree of 2. Quest uses the optimized FlashInfer kernels with block size 16 and top-K 64. InfLLM-v2 uses its original CUDA kernels with block size 64 and top-K 64. SeerAttention-R uses Qwen3-14B with block size 64, token budget 4096, and top-K 64. For every baseline, we adopt the configuration published by its authors rather than re-tuning it in our favor, and both the baseline and PRR run the same model, precision, block size, and cache budget, so that the only difference is the execution mechanism.

A.2 Extend Quest to GQA

The original Quest formulation targets Multi-Head Attention, where each query head has a dedicated KV head and independently selects its top-K pages. Under Grouped-Query Attention (GQA), G query heads share one KV head, making per-query-head selection both wasteful—selected pages are loaded once per KV head—and ill-defined when heads in a group disagree. We extend Quest to GQA by operating at KV-head granularity: page metadata (channel-wise min/max keys m_p, M_p) is stored per KV head as before, and for each KV head we form a representative query $\bar{Q} = \frac{1}{G}\sum_{g=1}^{G} Q_g$. Criticality estimation then proceeds as $s_p = \sum_i \max(\bar{Q}_i m_{p,i}, \bar{Q}_i M_{p,i})$, yielding one score per (KV-head, page) from which top-K pages are selected and shared across the group during sparse attention.

A.3 Temporal Locality across LLMs and Benchmarks

We measure how many blocks are repeatedly selected in consecutive decoding steps on five representative long-context benchmarks, including LongBench (Bai et al., 2024), InfiniteBench (Zhang et al., 2024), AIME (Hugging Face H4, 2025; Balunović et al., 2025), MATH500 (Lightman et al., 2023), and RULER (Hsieh et al., 2024) for GLM-Z1-9B (Team GLM, 2024), Deepseek-R1-8B (DeepSeek-AI, 2025), Llama3-8B-1M (AI@Meta, 2024), Qwen3-14B (Team, 2025), and Qwen3-32B (Team, 2025).

Table 8 reports the profiled overlap rates. Across LLMs and benchmarks, the overlap remains well short of the near-exact coverage that lossless reuse would require. This suggests that naively reusing the previous step's block selection offers little room for speculative speedup, due to frequently triggered fallback attention recomputation.

Table 8: Average overlap rate of blocks in consecutive block selections of LLMs under various benchmarks.

Model	Long Bench	Infinite Bench	RULER	AIME	MATH 500	Avg.
Quest						
GLM-4	0.669	0.694	0.658	0.674	0.664	**0.672**
GLM-Z1	0.689	0.693	0.700	0.695	0.707	**0.697**
DeepSeek-R1	0.629	0.639	0.638	0.633	0.639	**0.636**
Llama3	0.666	0.649	0.665	0.671	0.689	**0.668**
Qwen3-8B	0.667	0.688	0.697	0.689	0.703	**0.689**
Qwen3-14B	0.640	0.655	0.662	0.652	0.657	**0.653**
Qwen3-32B	0.637	0.657	0.640	0.655	0.633	**0.644**
InfLLM-v2						
GLM-4	0.689	0.701	0.679	0.718	0.702	**0.698**
GLM-Z1	0.692	0.708	0.701	0.698	0.694	**0.698**
DeepSeek-R1	0.669	0.657	0.681	0.670	0.677	**0.671**
Llama3	0.679	0.639	0.672	0.695	0.699	**0.677**
Qwen3-8B	0.681	0.662	0.682	0.714	0.713	**0.690**
Qwen3-14B	0.651	0.656	0.675	0.668	0.682	**0.666**
Qwen3-32B	0.646	0.642	0.663	0.644	0.680	**0.655**
SeerAttention-R						
Qwen3-8B	0.666	0.638	0.659	0.726	0.721	**0.682**
Qwen3-14B	0.672	0.633	0.643	0.726	0.721	**0.679**
DeepSeek-R1	0.673	0.654	0.710	0.715	0.716	**0.694**

A.4 Low Utilization of DSA

We profile the utilization of streaming multiprocessors (SMs), L2 bandwidth, and DRAM bandwidth for Qwen3-8B at larger batch sizes (i.e., 2, 4, 8, 16) under Quest, InfLLM-v2, and SeerAttention-R. The total context length is fixed at 128K, so batch size 16 corresponds to a per-prompt length of 8K tokens — well above the token budgets used by all three mechanisms. As shown in Table 9, GPU resources remain underutilized even at these larger batch sizes, leaving ample headroom for speculative attention to execute in parallel with block selection.

Table 9: Utilization (%) of SM, L2 bandwidth, and DRAM bandwidth under varying batch sizes for Qwen3-8B under Quest, InfLLM-v2, and SeerAttention-R.

DSA	Metric	BS=2	BS=4	BS=8	BS=16
Quest	SM	9.04	9.47	10.55	11.39
	L2 BW	39.81	40.43	42.77	44.80
	DRAM BW	33.63	34.20	36.02	37.58
InfLLM-v2	SM	7.10	7.86	9.48	12.56
	L2 BW	42.37	45.19	48.85	53.98
	DRAM BW	36.26	38.80	41.89	45.62
SeerAttention-R	SM	6.58	6.89	6.82	9.66
	L2 BW	31.94	33.27	32.28	42.71
	DRAM BW	23.38	24.24	22.85	30.43

A.5 Generality of the EMA Predictor

The predictor rests on a single property: the temporal stability of per-block importance across consecutive decoding steps. This property is architecture-agnostic, and the predictor consumes only the per-block importance scores that DSA already produces, making no assumption about model architecture, head layout, sparsity configuration, or the selection rule itself. Its smoothing hyperparameters are calibrated per prompt from prefill scores (§4.2), so it adapts to each model and configuration without manual tuning. Empirically, the predictor sustains a $\sim$98% overlap rate across seven LLMs from four families (§A.10). It also transfers to *trained* block selectors: on SeerAttention-R (Gao et al., 2026), whose gate is learned rather than heuristic, the same predictor attains 96.1% hit rate on average in §A.10, indicating that the mechanism extends to future DSA variants rather than being specific to the training-free ones we evaluate.

We further evaluate EMA's block-prediction recall for BABILong (Kuratov et al., 2024) and LooGLE (Li et al., 2024), two benchmarks that probe reasoning over information distributed across long contexts. As shown in Table 10, the predictor achieves hit rates ranging from 92.5% to 97.11% across DSAs and models. These results indicate that the EMA predictor can anticipate a large fraction of the blocks selected during decoding on these complex, inter-dependent reasoning workloads.

A.6 Search and Prediction Cost

Each grid evaluation runs the predictor over the prefill score matrix at a cost of $O(T_p \cdot N_p \log K)$ arithmetic operations. Crucially, the search is scheduled to overlap with the prefill phase: by the time the last prefill token is processed and IS is complete, most grid evaluations have already executed concurrently on the CPU. The critical-path cost of calibration is therefore the tail of the grid that cannot be overlapped. End-to-end, the full search—covering both overlapped and non-overlapped portions—takes approximately 2.4 ms and 11.2 ms for 16K- and 64K-token prefills, respectively, on GLM-4-9B. The non-overlapped portion (i.e., the fit for the final layer) accounts for only 0.06 ms and 0.28 ms for the two prefills. The prediction process takes only 0.01 ms.

A.7 Profiling Overheads

Profiling the execution time of block selection and speculative attention computation across context lengths (4K, 8K, 16K, 32K, and 64K) takes under 5 minutes on a single H100 GPU for GLM-4-9B. For context lengths outside this grid, we estimate execution time via polynomial regression, a standard technique for continuous resource modeling in LLM serving (Wang et al., 2025; Stojkovic et al., 2025; Hu et al., 2025).

A.8 Runtime Adjustment Over δ

Let A be the set of blocks chosen by the ground-truth top-K selection in the current round, and P denote the predicted blocks. The budget ratio δ bounds how many blocks the speculator is allowed to fetch. As shown in Figure 5, increasing δ admits more candidate blocks and tends to reduce misses,

Table 10: Average overlap rate of blocks in consecutive block selections of LLMs under complex distributed fact chaining benchmarks, BABILong and LooGLE.

Model	BABILong	LooGLE	Avg.
Quest			
GLM-4	93.51	91.53	**92.52**
GLM-Z1	94.67	93.41	**94.04**
DeepSeek-R1-8B	92.36	92.51	**92.44**
Llama3	95.34	93.00	**94.17**
Qwen3-8B	93.46	93.68	**93.57**
Qwen3-14B	94.27	93.96	**94.11**
Qwen3-32B	93.11	92.14	**92.63**
DeepSeek-R1-14B	93.60	92.00	**92.80**
InfLLM-v2			
GLM-4	97.20	95.06	**96.13**
GLM-Z1	97.89	96.33	**97.11**
DeepSeek-R1-8B	94.85	91.97	**93.41**
Llama3	97.57	94.94	**96.25**
Qwen3-8B	96.99	93.60	**95.30**
Qwen3-14B	96.89	94.60	**95.75**
Qwen3-32B	96.80	94.00	**95.40**
DeepSeek-R1-14B	97.11	95.21	**96.16**
SeerAttention-R			
Qwen3-8B	96.57	95.88	**96.23**
Qwen3-14B	96.44	95.74	**96.09**
DeepSeek-R1	96.50	96.22	**96.36**

but it also raises bandwidth pressure and the computational cost of speculative attention. Conversely, a smaller δ leaves more blocks unresolved on the critical path, capping the achievable speedup. The budget ratio δ must therefore be tuned carefully—large enough to cover as many blocks as possible for speculative attention, yet small enough that the speculation overhead does not itself extend the critical path.

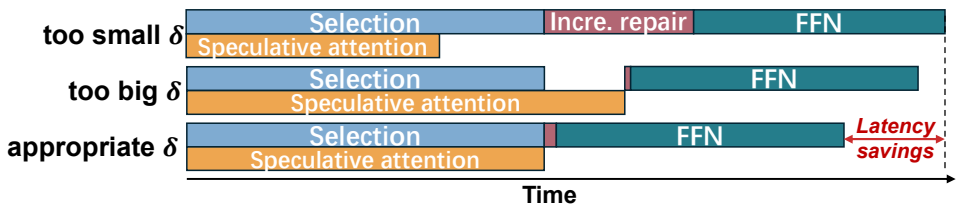

Figure 5: Impact of the speculation budget ratio δ on PRR's execution timeline. (top): When δ is too small, the predicted set covers too few true top-K blocks, leaving a large incremental repair stage on the critical path. (middle): When δ is too large, speculative attention itself exceeds the top-K selection window and delays the FFN. (bottom): An appropriate δ balances coverage and speculative cost, allowing speculative attention to overlap with top-K selection without delaying each other while leaving only a small repair stage before FFN.

A.9 Ablation Study

Table 11 decomposes PRR's end-to-end speedup into three cumulative stages (S1, S2, S3), evaluated across all six LLMs and five benchmarks under Quest and InfLLM-v2, and on a representative subset of models under SeerAttention-R, where S3 represents the standard PRR configuration.

We make three observations. First, naively enabling speculative attention through Stage-1 yields negligible improvement over serial DSA execution (1.00–1.04× across all configurations). Although the block selections reused from the previous decoding step overlap with the current true top-K set

by roughly 70% on average, Stage-1 requires *exact* coverage to be useful: a single missed block triggers a full attention recomputation over the true top-K selection. Since complete overlap is rarely achieved in practice, the fallback recomputation negates the latency savings from speculation. Second, Stage-2 introduces our customized sparse attention kernel, raising the average speedup to roughly $1.20\times$ under Quest and 1.20–$1.32\times$ under InfLLM-v2. Beyond simply accelerating attention over non-contiguous blocks, the kernel enables incremental attention repair, so missed blocks can be patched in directly rather than triggering a costly full recomputation. The gain is larger under InfLLM-v2 because its sparse-attention stage carries more latency, leaving more headroom for the repair path to exploit. Third, Stage-3 additionally activates the EMA-based predictor, which attains a 98% overlap rate between the predicted and true top-K sets (Table 12). This near-exact prediction sharply reduces the fraction of blocks that fall back to repair, lifting the average speedup to 1.28–$1.42\times$ under Quest and 1.32–$1.56\times$ under InfLLM-v2. The consistency of this stage-wise pattern across all six LLMs and five benchmarks indicates that PRR's design generalizes rather than being tuned to any single configuration.

Table 11: Ablation of decoding speedup across stages, models, and benchmarks. S1, S2, and S3 (PRR) are reported relative to serial DSA execution ($1.00\times$). Higher is better.

	LongBench			**InfiniteBench**			**RULER**			**AIME**			**MATH500**			**Avg.**		
Model	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3
Quest																		
GLM-4	1.00	1.27	1.50	1.01	1.21	1.40	1.00	1.24	1.45	1.00	1.11	1.30	1.00	1.16	1.42	1.00	1.20	**1.41**
GLM-Z1	1.00	1.23	1.48	1.01	1.20	1.39	1.00	1.22	1.43	1.00	1.15	1.33	1.00	1.20	1.45	1.00	1.20	**1.42**
DeepSeek-R1	1.00	1.19	1.39	1.03	1.21	1.42	1.00	1.25	1.47	1.00	1.13	1.35	1.00	1.13	1.39	1.01	1.18	**1.40**
Llama3	1.00	1.17	1.36	1.01	1.20	1.40	1.00	1.20	1.41	1.00	1.15	1.39	1.00	1.11	1.34	1.00	1.17	**1.38**
Qwen3-8B	1.00	1.19	1.40	1.00	1.18	1.37	1.02	1.20	1.43	1.01	1.22	1.41	1.01	1.18	1.40	1.01	1.19	**1.41**
Qwen3-14B	1.00	1.23	1.44	1.00	1.09	1.27	1.00	1.19	1.41	1.00	1.21	1.44	1.00	1.21	1.45	1.00	1.19	**1.40**
Qwen3-32B	1.00	1.06	1.26	1.00	1.15	1.34	1.00	1.10	1.27	1.00	1.08	1.27	1.00	1.05	1.26	1.00	1.09	**1.28**
InfLLM-v2																		
GLM-4	1.02	1.34	1.64	1.00	1.19	1.38	1.01	1.34	1.58	1.02	1.34	1.62	1.01	1.31	1.59	1.01	1.30	**1.56**
GLM-Z1	1.03	1.35	1.61	1.00	1.17	1.35	1.02	1.37	1.61	1.04	1.39	1.64	1.00	1.28	1.55	1.02	1.31	**1.55**
DeepSeek-R1	1.00	1.08	1.30	1.04	1.28	1.48	1.03	1.37	1.61	1.00	1.29	1.52	1.00	1.06	1.28	1.01	1.22	**1.44**
Llama3	1.00	1.12	1.32	1.04	1.26	1.45	1.00	1.31	1.55	1.00	1.30	1.55	1.00	1.07	1.30	1.01	1.21	**1.43**
Qwen3-8B	1.00	1.25	1.48	1.01	1.20	1.39	1.00	1.27	1.51	1.02	1.25	1.52	1.01	1.24	1.47	1.01	1.24	**1.48**
Qwen3-14B	1.00	1.31	1.56	1.03	1.23	1.43	1.00	1.28	1.51	1.00	1.29	1.53	1.00	1.27	1.53	1.01	1.28	**1.51**
Qwen3-32B	1.00	1.11	1.35	1.00	1.11	1.27	1.00	1.09	1.29	1.00	1.12	1.32	1.00	1.13	1.35	1.00	1.11	**1.32**
SeerAttention-R																		
Qwen3-8B	1.01	1.23	1.44	1.00	1.19	1.40	1.00	1.22	1.42	1.01	1.23	1.48	1.01	1.25	1.49	1.00	1.22	**1.44**
Qwen3-14B	1.01	1.17	1.42	1.00	1.15	1.33	1.02	1.19	1.44	1.02	1.23	1.43	1.01	1.21	1.43	1.01	1.19	**1.41**
DeepSeek-R1	1.01	1.18	1.38	1.02	1.23	1.41	1.01	1.23	1.44	1.02	1.24	1.42	1.01	1.21	1.43	1.01	1.22	**1.42**

A.10 EMA Hit Rate

Table 12 reports the overlap rate between EMA's predicted block set and the true top-K set of LLMs across benchmarks. PRR's EMA predictor consistently achieves a high overlap rate, exceeding 97% under Quest and InfLLM-v2 and 95% under SeerAttention-R.

A.11 Downstream Task Accuracy

Table 13 lists the downstream task score of PRR minus that of the corresponding DSA baseline. As one can observe, PRR doesn't degrade performance because PRR never drops important blocks, and it occasionally improves slightly when extra included blocks are helpful.

A.12 Isolating the Kernel from the Execution Mechanism

Table 12: The overlap rate(%) between EMA's prediction and the true top-K set of LLMs across benchmarks.

Model	Long Bench	Infinite Bench	RULER	AIME	MATH 500	Avg.
Quest						
GLM-4	98.65	96.86	98.36	98.15	98.25	**98.05**
GLM-Z1	98.55	97.33	97.95	97.93	98.09	**97.97**
DeepSeek-R1	98.56	97.60	96.99	97.71	97.88	**97.75**
Llama3	98.63	97.62	97.49	98.01	98.28	**98.01**
Qwen3-8B	98.46	98.04	98.69	97.83	98.24	**98.25**
Qwen3-14B	97.49	98.91	98.17	97.93	98.10	**98.12**
Qwen3-32B	98.52	98.56	98.12	97.94	98.06	**98.24**
InfLLM-v2						
GLM-4	97.86	97.14	96.46	98.15	98.01	**97.52**
GLM-Z1	99.12	98.89	98.56	98.42	98.49	**98.70**
DeepSeek-R1	99.03	99.01	98.27	99.40	98.57	**98.86**
Llama3	97.67	97.50	97.85	98.75	98.95	**98.14**
Qwen3-8B	98.27	97.34	98.42	97.64	98.14	**97.96**
Qwen3-14B	98.37	98.63	98.66	98.56	98.62	**98.57**
Qwen3-32B	98.87	98.80	98.93	98.82	98.87	**98.86**
SeerAttention-R						
Qwen3-8B	96.16	93.77	95.70	98.77	98.46	**96.57**
Qwen3-14B	95.05	92.59	93.19	98.16	98.06	**95.41**
DeepSeek-R1	95.43	93.89	96.73	97.71	97.91	**96.33**

Table 14: Decoding speedup of the serial baseline with PRR's optimized kernel and PRR on Qwen3-8B.

	Quest		InfLLM-v2		SeerAttn	
Bench.	**Opt.**	**PRR**	**Opt.**	**PRR**	**Opt.**	**PRR**
LongBench	1.13×	1.40×	1.23×	1.48×	1.18×	1.44×
InfiniteBench	1.11×	1.37×	1.22×	1.39×	1.16×	1.40×
RULER	1.15×	1.43×	1.26×	1.51×	1.17×	1.42×
AIME	1.13×	1.41×	1.26×	1.52×	1.22×	1.48×
MATH500	1.12×	1.40×	1.22×	1.47×	1.21×	1.49×
Avg.	**1.13×**	**1.41×**	**1.24×**	**1.48×**	**1.19×**	**1.44×**

To rule out our kernel itself as the source of the gain, we build same-kernel serial baselines: the optimized kernel is dropped into Quest, InfLLM-v2, and SeerAttention-R, which still run the original top-K selection and then attend over the full selected set—no prediction, speculation, reuse, or repair—so the gap to PRR isolates the benefit of breaking the selection-to-attention dependency. Normalized to each default DSA baseline on Qwen3-8B. As shown in Table 14, the kernel alone gives $1.13\times/1.24\times/1.19\times$ under Quest/InfLLM-v2/SeerAttention-R, while PRR reaches $1.41\times/1.48\times/1.44\times$. Predict-reuse-repair adds a further $1.25\times/1.20\times/1.21\times$ on top of optimized baselines, confirming the gain is not merely a kernel artifact.

A.13 Integration with Serving Engines

PRR fits engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) as a drop-in attention backend plus a small per-request predictor state. The EMA predictor is owned by the attention backend, and speculation and repair are both invoked inside the backend's forward pass, so no changes to the scheduler are required. The only change outside the backend is in the KV-cache

Table 13: Downstream task score of PRR minus that of the corresponding DSA baseline. Positive means PRR is better. $+0.00$ means no measurable difference.

Model	Long Bench	Infinite Bench	RULER	AIME	MATH 500
PRR − Quest					
GLM-4	+0.00	+0.01	+0.00	+0.00	+0.00
GLM-Z1	+0.00	+0.00	+0.00	+0.00	+0.00
DeepSeek-R1	+0.00	+0.00	+0.00	+0.00	+0.00
Llama3-8B	+0.00	+0.01	+0.00	+0.00	+0.00
Qwen3-8B	+0.01	+0.00	+0.00	+0.02	+0.00
Qwen3-14B	+0.00	+0.00	+0.00	+0.00	+0.00
Qwen3-32B	+0.00	+0.02	+0.00	+0.00	+0.00
PRR − InfLLM-v2					
GLM-4	+0.01	+0.00	+0.00	+0.00	+0.00
GLM-Z1	+0.00	+0.00	+0.00	+0.00	+0.00
DeepSeek-R1	+0.00	+0.01	+0.00	+0.00	+0.00
Llama3-8B	+0.00	+0.01	+0.00	+0.00	+0.00
Qwen3-8B	+0.00	+0.00	+0.00	+0.00	+0.01
Qwen3-14B	+0.00	+0.00	+0.00	+0.00	+0.00
Qwen3-32B	+0.00	+0.01	+0.00	+0.00	+0.00
PRR − SeerAttention-R					
Qwen3-8B	+0.00	+0.00	+0.00	+0.01	+0.02
Qwen3-14B	+0.00	+0.00	+0.00	+0.01	+0.00
DeepSeek-R1	+0.01	+0.00	+0.00	+0.00	+0.00

manager, which must expose a block-granularity mapping from engine KV-cache blocks to DSA KV blocks (the two use different block sizes in general).

A.14 Output Equivalence and Portability

Output equivalence after repair. On Qwen3-8B, the mean absolute error between PRR's repaired attention output and the corresponding standard DSA output is less than 0.007 across the five benchmarks, and 0.008 under SeerAttention-R. As analyzed in §4.3, this residual arises from retaining $P \setminus A$, not from the repair arithmetic.

Where Hopper is used. The repair kernel is a FlashAttention-based kernel that takes the previous pass"s output and log-sum-exp as the initial online-softmax state and continues attending over additional KV blocks. The only change relative to stock FlashAttention is exposing (and accepting) the per-pass log-sum-exp so that an attention pass can be resumed. It therefore reuses the standard FlashAttention datapath, inherits whatever backend the target GPU already provides, and introduces no Hopper-specific dependency. Hopper features are used only in the *speculative sparse-attention* compute path, for two optimizations: (i) loading the selected non-contiguous KV blocks directly, without first staging them in a contiguous buffer, and (ii) keeping the critical path free of FlashInfer"s split-K work partition and workspace layout. On architectures lacking these features, PRR falls back to a conventional gather-then-attend path; the predict-reuse-repair mechanism is unaffected.
Portability beyond Hopper. To isolate the repair kernel from the residual above and to check that it is not tied to our H100 testbed, we run a numerical-equivalence experiment on an A100 (Ampere) under `bf16`. We partition 64 KV blocks into two groups of 32, compute sparse attention over the first group, then fold in the second group with the incremental repair kernel, and compare against a single FlashAttention-2 forward pass over all 64 blocks. The maximum absolute difference is $3.74\text{e}{-4}$ at block size 64 and $2.42\text{e}{-4}$ at block size 128, on the order of `bf16` rounding error and not attributable to the repair itself. The repair kernel therefore reproduces the reference result on a non-Hopper architecture; only the speculative sparse-attention compute path uses Hopper-specific optimizations (§4.3).

A.15 KV Compression Related Works

PRR's key contribution is on breaking the selection-attention dependency, whereas these KV-retrieval and heterogeneous serving systems still follow the standard DSA's selection-attention execution flow: block selection -> block fetching from CPU -> attention computation. These methods instead optimize the fetching stage. For instance:

ParisKV (Qi et al., 2026) (a representative KV-retrieval system) pre-fetches related KV blocks from CPU to hide transfer latency by overlapping selection with fetching. ShadowKV (Sun et al., 2024) (a representative heterogeneous serving system) trades compute for data transfer, recomputing the full K cache from low-rank K vectors while V stays on CPU. In summary, these systems break the selection-fetching dependency (hide/reduce data-movement cost), while PRR breaks the selection-attention dependency (overlapping selection with speculative attention). These are two orthogonal optimization techniques but complementary to each other.