SparDA

Sparse Decoupled Attention for Efficient Long-Context LLM Inference

Paper authors: Yaosheng Fu, Guangxuan Xiao, Xin Dong, Song Han, and Oreste Villa

Paper discussion | 20 October 2026
arXiv:2606.04511v1 (3 June 2026)

Predict the next layer's KV accesses early enough to overlap transfer and compute.

Sparse attention leaves two expensive dependencies

KV capacity	The historical cache still grows with context length.
CPU–GPU transfer	Offloaded blocks must arrive before attention can run.
Block selection	Each query still scores a growing compressed-key index.

$$Q_l \longrightarrow \text{select blocks} \longrightarrow \text{fetch KV} \longrightarrow \text{attention}_l$$

The opportunity

Produce the memory-access decision before the layer needs its data.

Paper, Secs. 1 and 3, PDF pp. 1–3.

InfLLM-V2 selects blocks through a compressed index

$$\mathcal{B}_l(i) = \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}}(i) \cup \mathcal{B}_{\text{topk}}(i)$$

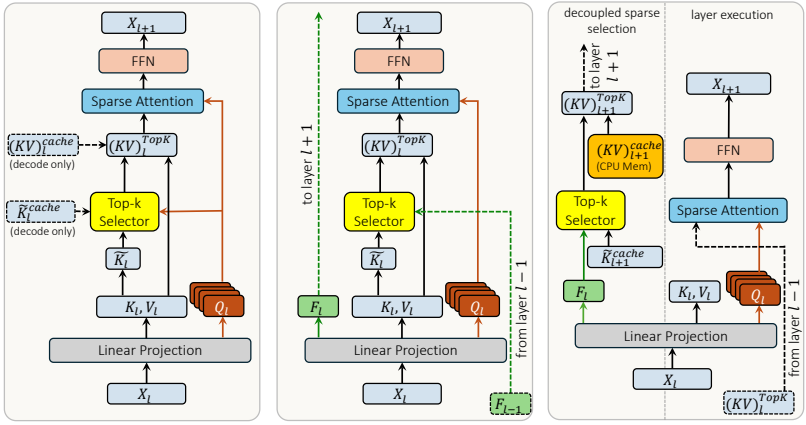
1. Mean-pool keys: window $l_{C_1} = 32$, stride $s_{C_1} = 16$.
2. Score with all G query heads in each GQA group; softmax per head, then sum scores within the group.
3. Max-pool scores to blocks, then select top-k.

Two granularities

A compressed-key position is an index entry.
A fetched KV block contains 64 tokens in these experiments.

Paper, Sec. 3.1, Eqs. (1)–(2), PDF p. 3; App. C, pp. 13–14.

Forecast separates selection from attention



(a) Baseline (InfLLM-V2) (b) SparDA (Prefill) (c) SparDA (Decode)

Original Fig. 2, PDF p. 4. Left: baseline; middle: prefill; right: decode.

Layer l predicts which history layer $l+1$ will read

$$(\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l, \mathbf{F}_l) = \phi_l(\mathbf{X}_l)$$

$$\mathcal{B}_{l+1} = \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}} \cup f_{\text{top}}\left(\mathbf{F}_l \widetilde{\mathbf{K}}_{l+1}^{\top}, k\right)$$

$$\mathbf{O}_{l+1} = \text{Attn}(\mathbf{Q}_{l+1}, \mathbf{K}_{l+1}[\mathcal{B}_{l+1}], \mathbf{V}_{l+1}[\mathcal{B}_{l+1}])$$

- Historical compressed keys for *every layer* already reside on GPU.
- Initial/local blocks are excluded from dynamic selection; the next layer's new key is unnecessary for selecting distant blocks.
- Layer 0 uses a same-layer Forecast and keeps its KV on GPU.

Paper, Eqs. (3)–(5), Sec. 4.1, PDF pp. 4–5; App. A, p. 13.

One Forecast head replaces G selection heads

Original selector	Forecast selector
G query heads per KV head	One Forecast head per KV head
Per-head softmax	Rank logits directly
Sum across query heads	Original attention heads retained

$$O(H_{\mathrm{kv}} G\, TN_b d_h) \longrightarrow O(H_{\mathrm{kv}}\, TN_b d_h), \qquad N_b \approx T/16$$

Prefill selection retains a quadratic length term.

Paper, Sec. 4.1, PDF p. 5. Complexity above is an explanatory derivation for scoring only.

Train Forecast to match the target layer's selector

$$\mathbf{S}^{\mathrm{tgt}}_{l,m} = \sum_{h=1}^{G} \mathrm{softmax}\Big(\mathbf{Q}_{l,m,h}(\widetilde{\mathbf{K}}^{\mathrm{tgt}}_{l,m})^{\top}/\tau\Big)$$

$$\mathbf{S}^{\mathrm{pred}}_{l,m} = \mathrm{softmax}\Big(\mathbf{F}_{l-1,m}(\widetilde{\mathbf{K}}^{\mathrm{pred}}_{l,m})^{\top}/\tau\Big)$$

- Teacher: actual target-layer query heads; student: previous-layer Forecast.
- Apply causal masking and initial/local exclusion.
- Match aligned score grids using a partitioned KL objective.
- Only 33.5M Forecast parameters train; the 8B backbone is frozen.

Paper, Sec. 4.2, Eq. (6), PDF pp. 5–6. m: GQA group; τ: attention temperature.

A rest bucket preserves supervision outside top-k

$$\mathcal{L}_{\mathrm{KL}} = \sum_{l} \mathrm{KL}\left(\bar{\mathbf{S}}^{\mathrm{tgt}}_{l,\mathcal{S}} \,\middle\|\, \bar{\mathbf{S}}^{\mathrm{pred}}_{l,\mathcal{S}}\right)$$

- Teacher-selected positions define the same partition for both branches.
- Keep k selected scores separately; merge all remaining scores.
- Normalize each resulting $(k+1)$-way distribution.

Teaching example, $k = 2$	Full distribution	Partition
Teacher	$(.40, .30, .20, .10)$	$(.40, .30, .30)$
Student	$(.35, .25, .15, .25)$	$(.35, .25, .40)$

Excess probability outside the selected set still incurs a loss.

Paper, Eq. (7), PDF pp. 5–6. Numerical example is explanatory, not an experiment.

Fine-grained supervision improves a coarse indexer

$$\underbrace{\text{Teacher keys: } (2,1)}_{\text{window, stride}} \longrightarrow \text{fine scores} \longrightarrow \text{max-pool to the } (32,16) \text{ grid}$$

$$\underbrace{\text{Student keys: } (32,16)}_{\text{same at training and inference}} \longrightarrow \text{student scores} \longrightarrow \text{partitioned KL}$$

Model	Teacher window	RULER	Four-family Avg
MiniCPM	$(32,16) \to (2,1)$	$75.7 \to 78.7$	$60.4 \to 61.7$
NOSA	$(32,16) \to (2,1)$	$71.4 \to 73.9$	$50.1 \to 51.7$

Strong local matches can survive before coarse key averaging blurs them.

Paper, Sec. 4.2, PDF p. 6; Table 6, p. 15. Higher scores are better.

A persistent UVA kernel turns lookahead into overlap

1. Compute current Q, K, V, F; update the current cache.
2. Forecast next-layer blocks; launch their prefetch on another stream.
3. Wait for current-layer data only if the prior prefetch is incomplete.
4. Run current attention and FFN while the next-layer transfer proceeds.

The system tradeoff

Persistent CTAs read pinned CPU memory via UVA.
More CTAs improve transfer throughput but consume compute resources.

$$t_{\text{transfer}} + t_{\text{compute}} \quad \rightsquigarrow \quad \max(t_{\text{transfer}}, t_{\text{compute}})$$

Paper, Sec. 4.3, PDF p. 6; Algorithm 2, p. 13. Timing expression is an idealized overlap illustration.

Evaluation distinguishes accuracy, latency, and capacity

- **Backbones:** MiniCPM4.1-8B and NOSA-8B; both use InfLLM-V2.
  NOSA additionally has a query-agnostic eviction head.
- **Accuracy:** HELMET, LongBench, RULER, and reasoning.
  Native contexts: 64K / 32K; RULER also extends to 128K.
- **Systems:** H100 80 GB / PCIe Gen5; A100 80 GB / Gen4.
  Both have 2 TB host memory; one warmup, two measured runs.

Read the baseline labels carefully

†: all KV on GPU. No †: CPU offload.
Decode comparisons at equal batch and at maximum feasible batch answer different questions.

Paper, Sec. 5.1, PDF pp. 6–7; App. C, pp. 13–15.

Average quality is preserved, with task-level tradeoffs

Model	Method	HELMET	LongBench	RULER	Reasoning	Avg
MiniCPM	Sparse	38.9	45.0	78.2	83.6	61.4
	SparDA	38.3	45.1	78.7	84.7	61.7
NOSA	Sparse	32.2	42.4	72.2	50.7	49.4
	SparDA	33.4	42.3	73.9	57.2	51.7

- Avg is the **unweighted mean of four benchmark families**.
- Dense averages remain higher: 63.5 / 52.4.
- InfiniGen averages: 57.7 / 45.6 under the block-sparse adaptation.
- MiniCPM loses 0.6 on HELMET; NOSA gains mostly on reasoning.

Paper, Table 1, PDF p. 7. All scores: higher is better.

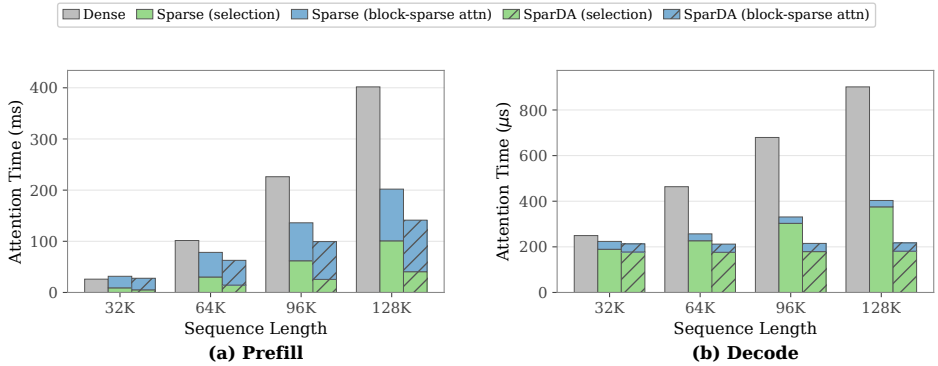
RULER improves at every tested length

Model	Method	32K	64K	96K	128K
MiniCPM	Sparse	86.1	78.2	68.7	67.7
	SparDA	87.6	78.7	70.8	68.8
	Gain (pp)	+1.5	+0.5	+2.1	+1.1
NOSA	Sparse	72.2	56.6	48.8	40.7
	SparDA	73.9	60.5	52.9	45.0
	Gain (pp)	+1.7	+3.9	+4.1	+4.3

- NOSA's relative advantage grows while its absolute accuracy falls.
- LongRoPE factors (MiniCPM) or RoPE-base scaling (NOSA) enable extension.
- Positional extension is identical across methods within each model.

Paper, Table 2, PDF p. 7; App. C, p. 14. pp = percentage points.

Selection savings appear in both inference phases



- At 128K: up to 2.50× faster prefill selection; over 2× in decode.
- Similar sparse-attention cost *between methods at the same length*.
- Near-flat decode latency reflects GPU underutilization in this regime.

Original Fig. 3, PDF p. 8; discussion p. 7. MiniCPM, batch 4. Units: ms (left), μs (right).

Prefill gains grow with context length

Model	Method	32K	128K
MiniCPM	Dense†	20388.3	8085.8
	Sparse	18548.4	13661.8
	SparDA	19845.6	17087.6
NOSA	Dense†	20438.9	8118.0
	Sparse	12778.3	9805.2
	SparDA	13456.0	11332.7

- H100, batch 4, tokens/s. † means no offload.
- At 128K: 1.25× / 1.16× versus Sparse.
- Prefill benefits mainly from cheaper selection; KV writeback is asynchronous.

Paper, Table 3 and Sec. 5.3, PDF p. 8.

At equal batch, decode reaches $1.69\times$ speedup

Model	Method	B4	B8	B16	B64
MiniCPM	Sparse	167.8	279.5	447.9	788.9
	SparDA	240.2	471.2	705.3	1000.1
NOSA	Sparse	173.2	285.4	529.2	1298.3
	SparDA	219.0	399.3	735.0	1463.3

$$\text{MiniCPM, B8: } \frac{471.2}{279.5} = 1.69, \qquad \text{NOSA, B8: } \frac{399.3}{285.4} = 1.40$$

- H100, 128K, tokens/s; both methods offload KV.
- NOSA already reduces fetch traffic, leaving less room for overlap.

Paper, Table 4, PDF p. 9; interpretation in Sec. 5.3, pp. 8–9.

The 5.28× result includes a larger feasible batch

MiniCPM, H100, 128K	Batch	Throughput (tokens/s)
Dense†	4	108.6
Sparse†	4	189.5
SparDA	64	1000.1

$$\frac{1000.1\ (\mathrm{B64})}{189.5\ (\mathrm{B4})} = 5.28\times \quad \text{versus Sparse}^{\dagger}$$

- Resident baselines run out of memory at B8 and above at 128K.
- This measures serving throughput across different feasible batches.
- When memory suffices, resident Sparse can win:
  at 32K/B16, 1073.0 versus SparDA's 918.5 tokens/s.

Paper, Table 4, PDF p. 9. †: no offload. Ratios calculated from the table.

High-batch gains depend on asynchronous prefetch

Model	Method	B4	B16	B64
MiniCPM	Sparse	167.8	447.9	788.9
	Forecast, no prefetch	247.3	505.9	696.9
	Full SparDA	240.2	705.3	1000.1
NOSA	Sparse	173.2	529.2	1298.3
	Forecast, no prefetch	226.5	653.0	1063.8
	Full SparDA	219.0	735.0	1463.3

- At B4, pipeline overhead slightly outweighs overlap.
- At B64, Forecast alone is slower than Sparse.
- Full prefetch adds 43.5% / 37.6% over the no-prefetch variant at B64.

Paper, Table 10, PDF p. 16. H100, 128K; tokens/s.

Prefetch needs a batch-dependent share of the GPU

Persistent CTAs	B16	B64
16	918.5	1255.9
32	813.1	1328.0
64	737.3	1380.2

- H100, MiniCPM, 32K; throughput in tokens/s.
- More transfer CTAs can slow attention/FFN by consuming SM resources.
- H100 heuristic: 16 CTAs below B32, otherwise 32.
  Within 4% of the best swept choice at every tested batch.
- A100 switches from 16 to 32 CTAs at B64.

Paper, Sec. 4.3, PDF p. 6; Table 7 and App. D, p. 15.

The same design also improves A100 throughput

Model	Phase / batch	Sparse	SparDA	Ratio
MiniCPM	Prefill / B4	6299.2	7756.8	1.23×
NOSA	Prefill / B4	4717.1	5477.5	1.16×
MiniCPM	Decode / B8	195.1	303.3	1.55×
NOSA	Decode / B16	395.6	526.3	1.33×

- All rows: A100 80 GB, 128K context, tokens/s.
- Decode gains are smaller than the H100 maxima.
- GPU compute, HBM, and PCIe all differ between platforms; these measurements do not isolate PCIe generation alone.

Paper, Tables 8–9, PDF pp. 15–16; hardware details p. 14. Last point: reader analysis.

Aggregate quality hides meaningful retrieval tradeoffs

MiniCPM: selected task changes
HELMET Recall: 67.8 → 61.5
HELMET Rerank: 14.7 → 17.0
RULER multi-key 3: 62.0 → 56.0
RULER QA 1: 64.0 → 70.0

NOSA: reasoning gains
MATH-500: 72.2 → 71.6
AIME 2024: 40.0 → 46.7
AIME 2025: 40.0 → 53.3
AIME has 30 problems per year.

AIME uses sampled generation; +13.3 percentage points is about four additional correct answers.

Paper, Tables 11, 13, 14, PDF pp. 17–18; evaluation protocol p. 14. Sparse → SparDA.

Early memory-access decisions are the central contribution

1. **Represent:** a compact Forecast predicts next-layer block demand.
2. **Teach:** fine-resolution supervision preserves useful selection quality.
3. **Schedule:** persistent UVA prefetch converts lookahead into overlap.

Where the evidence applies

Two InfLLM-V2-based 8B models; benefits depend on context and batch.
GPU memory still grows through compressed keys, layer-0 KV, and buffers.
Token-level DSA/CSA and larger models are proposed extensions.

Discussion: How far ahead should a model predict memory accesses before prediction error outweighs a longer overlap window?

Paper, Secs. 6–7, PDF pp. 9–10; App. C, pp. 14–15. Discussion question: reader analysis.

Fu et al., SparDA. arXiv:2606.04511v1.

Backup: training is small in parameters, substantial in compute

Trainable parameters	33.5M Forecast weights, 0.41% of the 8B model
Optimizer	AdamW; $\beta = (0.9, 0.95)$, $\epsilon = 10^{-6}$
Schedule	Constant 5×10^{-4}; 2000 optimizer steps
Batch / precision	Effective batch 32; BF16 mixed precision
Regularization	Weight decay 0.01; gradient norm clipped to 0.5
Data	ProLong-64K
Hardware	32 H100 GPUs

- Prose reports MiniCPM at 64K in $\leq$48 h; NOSA at 32K in $\leq$24 h.
- Table 5 lists 65,536 tokens without a model-specific distinction. NOSA's training length needs clarification for reproduction.

Paper, Sec. 4.2, PDF p. 5; App. B and Table 5, p. 13.

Backup: benchmark aggregation and generation protocols

- **HELMET:** 7 categories, 100 samples each; greedy decoding.
- **LongBench:** 14 English tasks; greedy decoding.
- **RULER:** 13 tasks, 50 examples each; greedy decoding.
- **Reasoning:** 500 MATH problems and 30 problems per AIME year; sampled generation, GPT-5.2 judging.

Reasoning differs across the two backbones

MiniCPM: thinking enabled, temperature 0.9, top-p 0.95, up to 64K output.
NOSA: two-shot examples, temperature 1.0, top-p 1.0, up to 8K output.

Four-family Avg weights each family equally; reasoning Avg weights each dataset equally.

Paper, App. C, PDF p. 14; Tables 1 and 14, pp. 7 and 18.

Backup: cache reuse and the first layer affect system gains

- NOSI caches blocks used in the previous decoding step.
  Transfer traffic comes from newly selected, missing blocks.
- GPU retains compressed keys ($\approx 1/16$ of full Key size),
  layer-0 KV, and working caches/buffers.
- At 96K, Sparse supports B128 while SparDA runs out of memory there;
  SparDA's B64 peak throughput still exceeds Sparse's B128 throughput.
- InfiniGen is adapted to block selection and performs CPU gather;
  its low throughput also reflects this implementation path.

Capacity and scheduling remain coupled

A larger lookahead window may hide more transfer, but also needs earlier predictions and more buffering.

Paper, Table 4, PDF p. 9; App. C, pp. 14–15. Final block: reader analysis.

Backup: prefill and decode use Forecast differently

Prefill

Compute the current layer's keys.

Score them with the stored $\mathbf{F}_{l-1}$.

Use the smaller selector on GPU.

Write used KV back to CPU asynchronously.

Decode

Next-layer historical keys already exist.

Score their GPU-resident compressed index using $\mathbf{F}_l$.

Fetch next-layer KV while computing the current layer.

Selection work still grows with history; flat measured decode latency is a hardware-utilization regime.

Paper, Sec. 4.1, PDF pp. 4–5; Algorithms 1–2, p. 13; Fig. 3, p. 8.