Open-Box: Punching Above the VRAM Weight Class

A reproducible research program for running larger effective models on small GPUs

Mark Bartlett
InsiderLLM mark@insiderllm.com
https://github.com/msb-msb/openbox-llm

July 2026

Abstract

Most "open" language models publish weights but not the means to rebuild them, and most efficiency research targets datacenter-scale hardware. We describe *Open-Box*, a research program with a single organizing thesis: *decouple a model's effective capability from what must reside in GPU VRAM*. Under this thesis we identify complementary bets—sparse attention, trainable memory layers, explicit hot/cold weight offload, and knowledge injection as an alternative to retrieval—each trading scarce VRAM for more abundant compute, host memory, or disk. As the program's first validated component, we present a clean-room, reproducible implementation of Native Sparse Attention (NSA) [1] with custom Triton kernels, gated for correctness against a reference at every construction step. We validate the kernels across attention configurations, confirm portability from Ampere (`sm_86`) to Hopper (`sm_90`) with all gates green, and report a +53% end-to-end training throughput improvement from a query-tiling optimization on the contiguous-range branches. We pair this with an honest negative result and a diagnosis: the same optimization does not transfer to the selection branch, and a per-branch profile shows selection is in fact the *dominant* training cost (52.6% of a step by profiler kernel-time, driven by its backward pass). A roofline analysis explains why—the selection branch is latency-bound, not compute-bound. Restructuring the selection backward block-outer and atomics-free, following concurrent NSA-kernel work [2], then makes it 8.4× faster at the 1.5 B shape, lifting the whole fwd+bwd step by a measured 1.08–1.36× that grows with sequence length and batch as the branch's wall-clock share grows—smaller than a naive profiler-share projection implies, and reported as a curve rather than a single number. We release weights, data, training code, and kernels under a permissive license. Our contribution is not the invention of NSA (due to DeepSeek-AI [1]) but its open, reproducible realization, cross-architecture validation, and honest performance characterization, positioned as the substrate for the broader program.

1 Introduction

Attention's $O(T^2)$ cost dominates long-context training and inference, and sparse attention has emerged as the leading remedy. Native Sparse Attention (NSA) [1] is a state-of-the-art instance: a natively-trainable design combining compression, selection, and sliding-window branches that matches full-attention quality while cutting FLOPs. Its efficiency, however, lives in hand-written kernels, and the open implementations of those kernels are typically tuned to one GPU generation and not validated across others—a gap that matters most to the practitioners least able to absorb it.

That audience is the one running models on a single consumer or workstation GPU, where the binding constraint is not compute but *VRAM*: a 24 GB card caps parameters, KV cache, and

activation memory alike. Datacenter-scale efficiency work rarely targets this regime. The practical goal is to make a small GPU behave like a larger one.

We present *Open-Box*, a research program organized around one thesis—*decouple a model's effective capability from what must reside in GPU VRAM*—and its first validated component. Each part of the program spends an abundant resource (compute, host RAM, disk, an explicit index) to relieve the scarce one (VRAM); we state the full set of bets, labeled by confidence, in §2. This paper's concrete contribution is a clean-room, reproducible NSA implementation whose Triton kernels are gated against a reference at every step and validated from Ampere (`sm_86`) to Hopper (`sm_90`). We report a +53% training-throughput optimization on the contiguous-range branches, an honest negative result on the selection branch, and a profile showing selection dominates the training step (52.6%)—a finding independently corroborated by concurrent NSA-kernel work [2]. Acting on that profile, we rebuild the selection backward block-outer and atomics-free, making it 8.4× faster in isolation and the whole fwd+bwd step a measured 1.08–1.36× faster (rising with sequence length and batch; §7). We claim no large trained model; our own throughput measurements (§7) show why one is a funded, not incidental, undertaking. We position Open-Box as an open, reproducible *substrate* and program, not a faster kernel than the specialized implementations [2, 3] we build on.

Contributions.

- A statement of the Open-Box research program and its thesis (§2), each bet labeled by how proven it currently is.
- A clean-room, reproducible NSA implementation built as a gated "rung ladder," correctness-checked against a reference at every step (§4–5).
- Cross-architecture validation from Ampere to Hopper, all gates green (§6).
- Two measured optimizations—a +53% query-tiling gain on the range branches and an 8.4× block-outer rebuild of the dominant selection backward (a measured 1.08–1.36× end-to-end fwd+bwd, rising with context)—with a roofline analysis and an honest negative result explaining which fix applies where (§7).
- Full release of weights, data, code, and kernels under a permissive license.

2 The Open-Box Program

The thesis—decouple capability from VRAM—is realized by a small architecture and a set of bets.

Architecture. A compact reproducible *base* model, a *growable memory store*, and *explicit-selection offload*. The base is small enough to understand and train on modest hardware. Capacity grows by adding memory entries rather than retraining. Because selection is explicit (a lookup, not a learned prediction), residency of weights across GPU / host / disk is tractable to schedule.

The bets. We label each by current confidence, because a research program stated honestly is more useful than one stated optimistically:

- **Native Sparse Attention (NSA)** [1] — *validated here.* Reduces attention's $O(T^2)$ cost via compression, selection, and sliding-window branches. The subject of this paper.

- **Trainable memory layers** [4] — *published, our bet to validate at scale.* Capacity in an addressable store rather than dense parameters; the mechanism by which the base "grows" without retraining.
- **Hot/cold offload** — *the mechanism that makes the small-GPU goal real.* Keep hot weights in VRAM, cold weights in host RAM or on disk, and swap by need. PowerInfer [5] *predicts* activation with an auxiliary model; explicit top-k memory selection instead makes "which weights are cold this token" a lookup rather than a forecast. Concurrent work on offloadable sparse attention (NOSA [3]) demonstrates the block-wise GPU/CPU swapping this bet depends on.
- **Knowledge injection vs. retrieval** — *most speculative.* Inject a document's knowledge directly into memory—editable and inspectable—rather than retrieving chunks into context. We build on Hay's LARQL [7], which decompiles a model into a queryable index and edits its knowledge without retraining; the open question is whether injected knowledge recalls *faithfully* at document scale versus a retrieval baseline. In our own prior experience the tooling (document $\rightarrow$ text) is trivial; faithful recall is the hard part.

All bets spend abundant resources to relieve VRAM. This paper validates the first and establishes the substrate the rest depend on.

3 Background: Native Sparse Attention

NSA [1] replaces dense attention with three branches whose outputs are gated and summed. For query q_t at position t, with keys/values partitioned into blocks of size B,

$$o_t = \sum_{c \in \{\text{cmp, sel, win}\}} g_t^c \, \text{Attn}\Big(q_t, \, \tilde{K}_t^c, \, \tilde{V}_t^c\Big), \qquad g_t^c \in [0,1], \tag{1}$$

where g_t^c are learned gates and $(\tilde{K}_t^c, \tilde{V}_t^c)$ are the keys/values each branch attends:

- **Compression.** Blocks are pooled into summaries $\tilde{K}_t^{\text{cmp}} = f_K(K_{\le t})$, giving cheap global coverage over $\lceil t/B \rceil$ entries.
- **Selection.** Block importance is scored from the compression branch, $p_t = \text{softmax}(q_t \tilde{K}_t^{\text{cmp}\top})$, and the top-$k$ blocks are gathered at full resolution: $\mathcal{I}_t = \text{Topk}(p_t)$, $\tilde{K}_t^{\text{sel}} = K[\mathcal{I}_t]$. Crucially $\mathcal{I}_t$ is computed under stopgrad—no gradient flows through the block choice. Preserving this (we call it *trap-1*) is a correctness invariant throughout.
- **Sliding window.** The most recent w tokens $\tilde{K}_t^{\text{win}} = K_{t-w:t}$ are attended densely for local structure.

Compression and window both attend a *contiguous causal range*; selection attends a *scattered, per-query* set of blocks—a distinction that turns out to govern which optimizations apply (§7). We use grouped-query attention (GQA) and implement the kernels in Triton [6].

4 Implementation

We build the implementation as a *rung ladder*: each rung is a self-contained step that must pass a correctness gate before the next begins, so regressions stay local and every claim is backed by a green gate.

1. **Pure-PyTorch NSA** — a readable reference implementation of all three branches.
2. **Selection forward kernel** — a Triton kernel for the selection branch's gather-and-attend, using an online softmax to avoid materializing the attention matrix, matched against the reference.
3. **Selection backward kernel** — gradients dQ, dK, dV, gated by gradient parity against autograd through the reference. Softmax statistics are handled explicitly; *trap-1* keeps block selection non-differentiable.
4. **Fusion** — compression and window, which both attend a contiguous causal range $[\text{lo}, \text{hi}]$, are unified into one range-attention kernel; the three branches are combined via autograd rather than a monolithic fused backward.
5. **Model integration** — the fused path is wired into the model behind a flag, defaulting to the reference path, so kernel-path training is opt-in and verifiable.

5 Validation

Correctness. Each kernel is gated by `allclose` against the reference across a configuration matrix spanning batch size, GQA group ($G \in \{1, 2, 3, 4\}$), head dimension ($D \in \{32, 64, 128\}$), sequence length, block size, and selection counts exceeding a row's valid block span. Forward, backward (via gradient parity), and the fused three-branch path all pass in both `fp32` and `bf16`, with maximum errors well inside tolerance (forward $\max_{\text{abs}} \approx 5.4 \times 10^{-7}$; backward worst-case $|\Delta g| \approx 2.5 \times 10^{-6}$ in `fp32` against a 3×10^{-3} gate).

Model-level parity. Wired into the full model, the kernel path matches the reference on both loss and per-parameter gradients, confirming the flag is a true drop-in that does not alter training dynamics.

Quality. On a matched-baseline comparison at 152.96 M parameters (`d768/L16`, GQA 12q/4kv) trained on FineWeb-Edu, NSA reaches a held-out validation loss of 5.44 versus 5.42 for full attention—a +0.53% gap, i.e. competitive quality alongside the memory benefit below. The kernel path tracks the reference step-for-step (Fig. 1).

Memory scaling. At the model level the fused path uses 7.77 GB versus 14.05 GB for dense at matched configuration (−44.7%), and remains resident at sequence lengths where dense runs out of memory (Fig. 2).

6 Cross-Architecture Portability

The kernels were developed and gated on an Ampere RTX 3090 (`sm_86`). We re-ran the full gate suite—forward, backward, fused, and model integration—on a Hopper H100 (`sm_90`) without source changes. The Triton autotuner re-tunes per device; no architecture-specific code was required. All gates passed in both `fp32` and `bf16`, establishing portability across two GPU generations and validating the implementation on the architecture intended for larger runs.

7 Performance Analysis

Query-tiling (+53% training). The range-attention kernel (compression and window) initially padded the GQA group to the hardware matrix-multiply row floor, underusing tensor cores. By

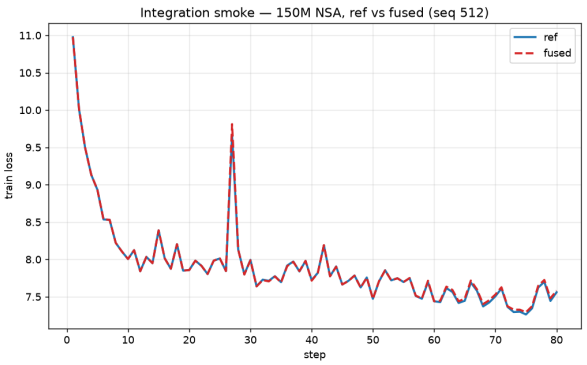

Figure 1: Model-level loss parity, 152.96 M params: the fused kernel path tracks the reference step-for-step, confirming a true drop-in.

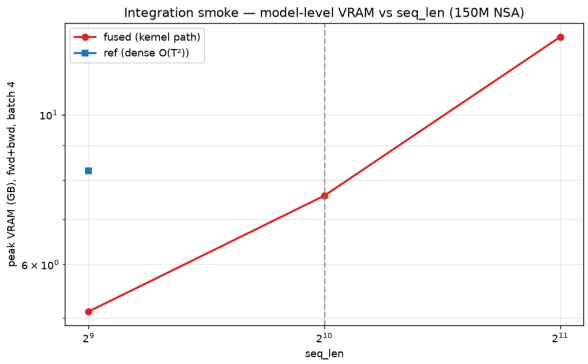

Figure 2: Model-level VRAM vs. sequence length: fused NSA stays resident where dense OOMs (−45%).

tiling multiple query blocks per program so adjacent queries share a contiguous key range, the MMA tiles fill with real work. This raised the fused *forward* microbenchmark by +46% at sequence length 2048 (the window branch alone improved 4.3–4.8×; the whole-forward figure is diluted by the unchanged selection branch), and lifted *end-to-end* training throughput by +53% (5,765 → 8,807 tokens/s, 198 M params, RTX 3090), with all correctness gates green (Fig. 6). Backward benefits as well as forward, and backward dominates training, so the end-to-end gain exceeds the forward microbenchmark.

An honest negative: selection does not tile. Applying the same query-tiling to the selection branch does not work, and the reason is structural. Unlike the range branches, selection performs a *per-query* top-k gather, so adjacent queries do not share a key range; stacking their blocks to fill the MMA merely moves padding from rows to columns while conserving total work. A roofline measurement confirms the branch is neither compute- nor bandwidth-bound—at a representative configuration it reaches only a few percent of peak on either axis. It is *latency/occupancy-bound*: many small programs each chasing a short chain of dependent gather loads through a masked online softmax. A flash-decode-style split adding parallelism produced no improvement (within noise), consistent with the kernel already saturating occupancy.

Where the cost actually is. A per-branch profile of a full training step at 1.5 B parameters (Fig. 5) shows selection is the *dominant* cost at 52.6% of the step—almost entirely its backward pass (45.0% backward vs. 7.6% forward). Dense feed-forward and projections account for 19.3%, the range branches 18.6%, and everything else 9.5%. Measured model FLOPs utilization is ∼32% against the RTX 3090's honest bf16 peak of ∼35 TFLOP/s (the card halves throughput under fp32 accumulation, so this peak—not the nominal 71 TFLOP/s—is the correct denominator). The optimization that worked addressed the range branches; the branch that dominates is precisely the one it cannot help. The route forward is not wider MMA tiles but reducing the selection backward's overhead.

Rebuilding the selection backward (8.4×). We restructured the selection backward along the lines of concurrent NSA-kernel work [2], in dependency order: (i) emit the log-sum-exp statistic from the forward pass and build a causal, block-to-query inverted index (CSR); (ii) rewrite dK/dV block-outer—one program owns a KV block, loads it once, and loops the real queries that selected it,

so the matrix-multiply tile fills without GQA padding and the scatter needs no atomics; (iii) make dQ block-outer as well, fusing $dq = ds\,K$ into the same pass and consuming the stored statistic in a single softmax pass. Each step preserved *trap-1* (the block index stays non-differentiable), and a shape gate retains the original kernel for the small configurations where the restructured path's fixed cost would not amortize. All correctness gates stayed green and kernel-vs-reference gradient parity held (fp32 $\lesssim 10^{-5}$). At the 1.5 B shape the selection backward drops from 23.3 to 3.03 ms (8.4×; 87.9k → 676k tok/s). Measured on the full fwd+bwd step (RTX 3090, fused path, no optimizer), swapping the atomic selection backward for the FSA kernel yields a *configuration-dependent* end-to-end speedup that rises with the branch's share of the step: 1.08× at seq 512, 1.18× at seq 1024, and 1.26× at seq 2048 (batch 1), reaching 1.36× at seq 1024/batch 2 (Fig. 3). The isolated 8.4× converts to these smaller end-to-end figures because the branch's *wall-clock* share (∼17–30% across these configs) is smaller than the 45% profiler *kernel-time* share reported above: profiler self-CUDA time excludes launch gaps and counts overlap, so it overstates how much of the step the branch actually occupies. Killing a minority of the step, however fast, bounds the whole-step gain accordingly. The speedup grows with context but saturates: as sequence length increases, the range branches and the dense projections grow too, so selection's share stops rising and the end-to-end gain plateaus. We confirm this on an H100 (80 GB), which reaches the long-context, batch-2 configurations the 3090 cannot hold: the measured speedup is 1.10× at seq 512, 1.22× at 1024, 1.29× at 2048, and 1.33× at 4096, then eases to 1.31× at 8192 as selection's share peaks near 28% and holds (Fig. 3). The H100 curve sits slightly below the 3090's at matched configurations—the Hopper's faster dense matmul shrinks selection's relative share, so the end-to-end gain is smaller—while the two track within a few points and their selection-share profiles nearly coincide, indicating the speedup is a property of the algorithm and configuration rather than the device. The kernels are portable to `sm_90` (all gates green) but not Hopper-tuned; the ratios we report are unaffected by absolute-throughput tuning, since they compare the same fused path with the selection backward on versus off. A naive projection that multiplies the isolated 8.4× by the 45% profiler share would overstate all of these measured end-to-end figures.

Selection forward. Applying the same block-outer restructuring to the selection *forward* (a split-K structure emitting per-query softmax partials into CSR-indexed buffers, combined by log-sum-exp) makes it 1.57× faster in isolation at the 1.5 B shape (4.28 → 2.40 ms; up to 2.4× at larger head dimension). Since the forward is only ∼7.6% of the step, the incremental end-to-end gain is small (1.034× over the backward-only result), bringing the combined kernel speedup to ∼1.69×. A stricter shape gate ($D \geq 128$) keeps the original path where the CSR overhead would not amortize.

8 Related Work

NSA [1] introduced the compression/selection/window design and hardware-aligned Triton kernels, validated at 27 B parameters. Our work is an independent, open reproduction at consumer scale with cross-architecture validation, not a novel mechanism.

Flash Sparse Attention (FSA) [2] is the closest peer: an alternative NSA-kernel implementation that inverts the vanilla loop order (KV blocks outer, query tokens inner) so a full tile of queries attends each KV block, eliminating the GQA-group padding that starves tensor cores when groups are small. It handles the resulting cross-block reduction with a decoupled online-softmax pre-computation kernel and a separate reduction kernel, avoiding atomics. FSA reports up to 3.5× (avg 1.6×) kernel-latency and up to 1.25× (avg 1.09×) end-to-end training speedups. Two points connect directly to us: FSA's padding diagnosis is the same one our query-tiling addresses for the

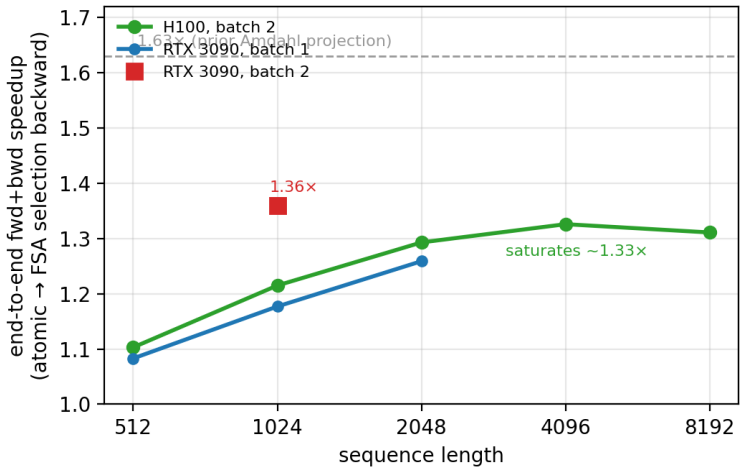

Figure 3: Measured end-to-end fwd+bwd speedup (atomic → FSA selection backward) versus sequence length at the 1.5 B shape, on an RTX 3090 (24 GB) and an H100 (80 GB). The gain rises with context as the selection backward's wall-clock share grows, then saturates near 1.33× once the range branches and dense projections also grow; the H100 reaches the long-context batch-2 configurations the 3090 cannot hold. Both cards track within a few points, and neither approaches the 1.63× a profiler-share Amdahl projection would imply (dashed).

range branches; and FSA's own breakdown finds token selection dominates attention cost (up to 79%, avg 65%), independently corroborating our 52.6% selection profile. We adopt their block-outer, atomics-free restructuring for our selection backward (§7), reaching 8.4× on that branch at the 1.5 B shape; we do not claim to match FSA's fully-tuned kernel, but the same design principles transfer to our open substrate.

NOSA [3] makes NSA-style selection natively offloadable, constraining CPU→GPU KV transfer via a locality-bounded, block-wise selection so that decoding stays communication-cheap—prior art for our hot/cold offload bet.

Relative to all three, our contribution is reproducibility and open release, cross-generation (`sm_86` → `sm_90`) validation, and situating NSA within a broader VRAM-decoupling program. We do not claim faster kernels than FSA; where speed is the goal, FSA's design is the one to adopt.

9 Limitations and Research Program

We deliberately claim no large trained model. Our throughput measurements show a compute-optimal 1.5 B run remains a datacenter-budget undertaking on rented hardware even after this speedup—the selection-backward rework lowers the cost but does not eliminate the need for funded compute. What we establish is the *substrate*: correct, portable, and—after the rework of the dominant branch—reasonably efficient NSA kernels, with a performance envelope characterized honestly and its worst cost identified and then removed.

Roadmap. The program proceeds by *component*, not by model size, each step gated by the last:

1. **Distillation from a larger open teacher** — with the base trained, distill from an open 27–35 B teacher (logit/KL) to lift a small-token-budget 1.5 B student to a usable capability level; the substrate the bets below are tested on.

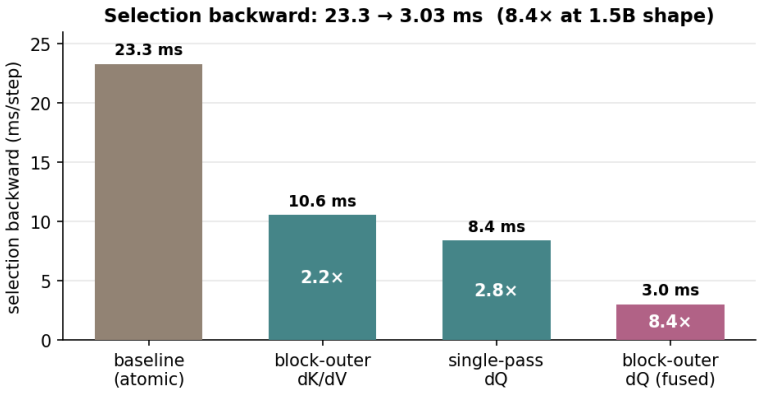

Figure 4: Rebuilding the selection backward in dependency order. At the 1.5 B shape the branch drops 23.3 → 3.03 ms (8.4×), all correctness gates green throughout.

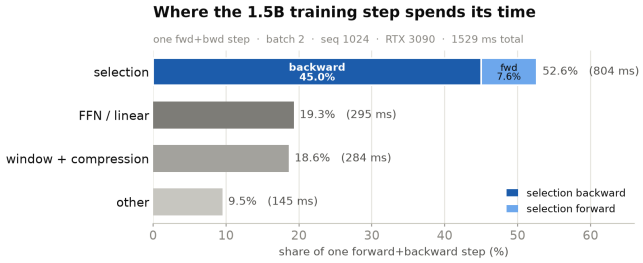

Figure 5: Per-branch share of a 1.5 B fwd+bwd training step. Selection dominates (52.6%), driven by its backward.

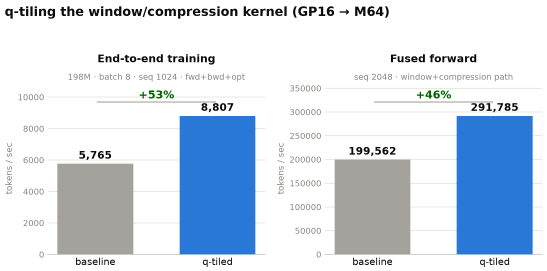

Figure 6: Query-tiling: end-to-end training +53% and fused-forward +46% (seq 2048).

2. **Trainable memory layers** [4] at the current scale — test that capacity added as memory entries competes with dense parameters.
3. **Hot/cold offload** — drive residency from explicit memory-layer top-k, building on offloadable-selection results [3].
4. **Knowledge injection vs. retrieval** — the most speculative bet, gated on the above.

Each is a component a validated substrate plus modest compute unlocks; larger trained models follow from funding, not from the roadmap itself. Everything needed to rebuild what is described here is public; we invite reproduction, forking, and collaboration.

10 Conclusion

Open-Box is an attempt to make small GPUs punch above their VRAM weight class, in the open and from the ground up. We stated the program's thesis and bets honestly, and validated its first component—a clean-room, reproducible NSA implementation—across two GPU architectures, with a measured +53% throughput optimization and a clearly reported negative result that also uncovered the true dominant cost. The substrate is real; the program is the work ahead.

Acknowledgments. We thank Chris Hay for LARQL [7], which informs the knowledge-injection bet.

References

[1] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. X. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, W. Zeng (DeepSeek-AI). *Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention.* arXiv:2502.11089, 2025.

[2] *Flash Sparse Attention: An Alternative Efficient Implementation of Native Sparse Attention Kernel.* arXiv:2508.18224, 2025.

[3] *NOSA: Native and Offloadable Sparse Attention.* arXiv:2510.13602, 2026.

[4] V.-P. Berges, B. Oğuz, D. Haziza, W.-t. Yih, L. Zettlemoyer, G. Ghosh. *Memory Layers at Scale.* arXiv:2412.09764, 2024.

[5] Y. Song, Z. Mi, H. Xie, H. Chen. *PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU.* arXiv:2312.12456, 2023.

[6] P. Tillet, H. T. Kung, D. Cox. *Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations.* MAPL, 2019.

[7] C. Hay. *LARQL: Querying and Editing Model Weights as a Graph Database.* https://github.com/chrishayuk/larql, 2025. Talk: *LLMs Are Databases—So Query Them*, https://www.youtube.com/watch?v=8Ppw8254nLI.