POSTER: SAIL: SRAM-Accelerated LLM Inference System with Lookup-Table-based GEMV

Jingyao Zhang
jzhan502@ucr.edu
University of California, Riverside
Riverside, California, USA

Jaewoo Park
hecate64@unist.ac.kr
Ulsan National Institute of Science and Technology
Ulsan, Republic of Korea

Jongeun Lee
jlee@unist.ac.kr
Ulsan National Institute of Science and Technology
Ulsan, Republic of Korea

Elaheh Sadredini
elahehs@ucr.edu
University of California, Riverside
Riverside, California, USA

Abstract

Efficient LLM inference on CPUs faces two challenges: weak support for the sub-8-bit arithmetic required by quantized models and severe memory bottlenecks during token generation. SAIL integrates lookup-table-based GEMV with near-cache SRAM processing-in-memory, tensor-level scheduling with ping-pong data movement, a pattern-aware LUT optimization that reduces cycles by 13.8%, and bit-exact in-memory type conversion. SAIL supports Q2–Q8 precision with about 2% hardware overhead and one ISA instruction. A calibrated gem5 evaluation shows up to 10.4× speedup and 19.9× higher tokens per dollar than ARM Neoverse-N1 CPUs, with competitive cost efficiency relative to V100 and A100 GPUs and no additional accuracy loss beyond quantization.

Keywords

processing-in-memory, large language model inference, lookup-table-based GEMV, near-cache computing, quantization

ACM Reference Format:
Jingyao Zhang, Jaewoo Park, Jongeun Lee, and Elaheh Sadredini. 2026. POSTER: SAIL: SRAM-Accelerated LLM Inference System with Lookup-Table-based GEMV. In *International Conference on Parallel Architectures and Compilation Techniques (PACT '26), October 19-22, 2026, Chicago, IL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3838684.3846829

1 Motivation and Key Insight

LLM inference has one compute-bound prefill phase and repeated memory-bound token generation. Generation streams weights for GEMV at every token. Quantization cuts traffic, but sub-8-bit formats still require unpacking and conversion and underuse CPU/GPU lanes in llama.cpp [4]. Flexible number formats and non-uniform quantization improve robustness and accuracy [6, 8]; SAIL targets the orthogonal execution and data-movement costs.

PIM reduces data movement, but bit-serial in-cache designs often require resident data [1, 3]. Fulcrum addresses practical in-situ control and access [7]; software LUT-GEMV still moves entries through the LLC [9]. SAIL maps each lookup to an SRAM row and co-locates LUTs with weights, avoiding full-matrix LLC-to-core traffic while supporting Q2–Q8.

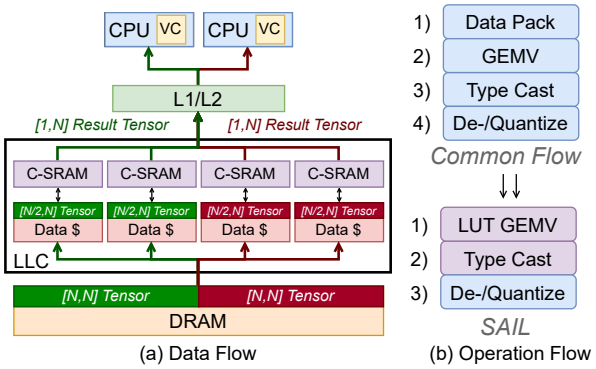


Figure 1: SAIL data flow. Weights remain on the DRAM–LLC–C-SRAM path; only result vectors return to the CPU.

2 Architecture and Execution

Each LLC slice adds a 256×512-bit C-SRAM, while a core-side Data Feeding Module broadcasts activation patterns and aggregates partial sums. C-SRAM contains an array, modified sense amplifiers, a transposer, and control. It adds cache capacity when idle; during SAIL it is transient scratch outside coherence, with results coherent at instruction completion.

C-SRAM constructs weight-combination LUTs online, and each activation pattern selects one row for accumulation. This avoids offline model expansion of up to 3.75× at Q4 with NBW=4. A `lutmm_1k` instruction computes a $[1, 1024] \times [1024, 1024]$ tile and specifies addresses, scale factors, and quantization level; larger GEMVs compose multiple tiles. Related work covers configurable PIM compilation and bit-parallel/bit-serial layout selection [16, 17].

Layer-wise ping-pong buffering overlaps the next DRAM load with current C-SRAM lookup and aggregation; the pipeline is optimal at batch size eight. Online LUT construction costs 3–12% of end-to-end cycles, while processing a request batch per layer amortizes each weight load. A 32-entry Pattern Reuse Table recognizes about 17% repeated patterns at batch eight, reducing compute cycles by 13.8%.

This work is licensed under a Creative Commons Attribution 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846829

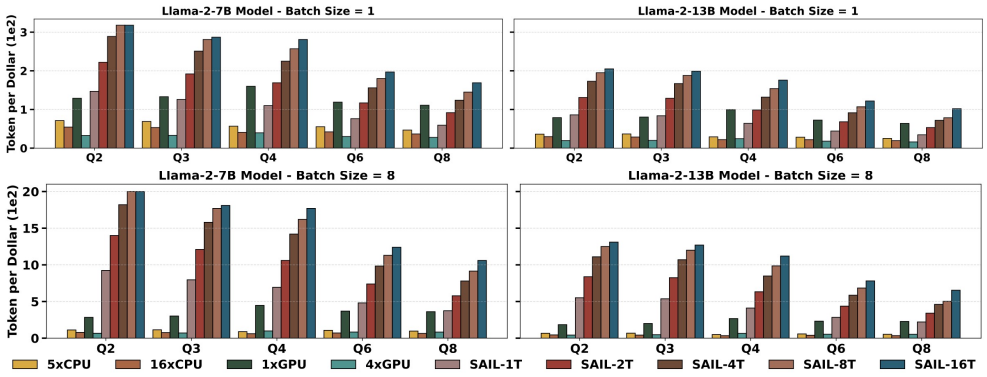


Figure 2: Tokens per dollar across platforms for Llama-2-7B/13B at batch sizes 1 and 8.

C-SRAM converts signed n-bit integers ($n \leq 25$) to FP32 before CPU scaling. All 2^{25} inputs match reference C; values exceeding 2^{24} use the CPU vector fallback. At 16 threads, 512 KB of C-SRAM adds 1.8% area and 2.2% power at 45 nm. Other work covers near-cache/in-SRAM cryptographic and memory-centric systems [10–15] and untrusted-PIM security [5].

3 Methodology and Results

We evaluate TinyMistral-248M and Llama-2-7B/13B at batch sizes 1/2/4/8, context length 4096, and Q2–Q8 using llama.cpp. ARM and SAIL run in cycle-level gem5 [2], calibrated within 5.4% of an ARM Neoverse-N1 server. C-SRAM counts are analytical and hardcoded; gem5 models NoC, DRAM, and cache effects. The 32-core, 3 GHz model has a 32 MB LLC, an 8×8 mesh, eight DDR4-3200 channels, and 32 C-SRAM arrays.

At 16 threads, SAIL reaches 81.63 and 26.25 tokens/s for 7B Q2 and 13B Q8, versus ARM's 9.30/4.80 and AMX's 24.96/10.07. It leads all CPU baselines at Q2/Q4/Q8; the complete Q2–Q8 sweep yields 4.8×–8.8× speedup for 7B and 5.5×–10.4× for 13B.

For Llama-2-7B Q4, incremental throughput is 9.85 tokens/s on ARM, 28.4 with LUT-GEMV, 53.1 with pipelining, 62.8 with in-memory conversion, and 72.1 with pattern reuse (7.32×). At batch 8 and 16 threads, latency is 68% FFN GEMV, 19% attention projections, 5% KV-cache work, 4% LUT construction, and 4% conversion/dequantization.

Figure 2 reports TPD: 30 days of token throughput divided by monthly on-demand GCP price (US-East, January 2026). At batch one, a V100 generally leads SAIL-1T except at Q2; at batch eight, SAIL exceeds single-GPU TPD at nearly all precisions. Peak TPD is 19.9× ARM. NoC utilization is 69% at 16 threads and 97% at 32. Because SAIL is bit-exact, accuracy follows GGML quantization: for 7B, Q4+ adds less than 0.5 perplexity over FP16, while Q2 adds 2–4 points.

4 Discussion and Conclusion

All speedups concern decode; ARM and SAIL are simulated under matched conditions with identical software. SAIL targets multi-user throughput, while latency-sensitive serving needs additional scheduling. Production GPU stacks could improve GPU throughput 2–4×, narrowing Q4 GPU–SAIL gaps to ~1.8–3.5×; continuous batching could also benefit SAIL.

SAIL places configurable C-SRAM beside the LLC for in-place LUT-GEMV and returns only $[1, N]$ results to the CPU. It adds one instruction without coherence-protocol changes. The calibrated evaluation shows 10.4× speedup and 19.9× higher TPD than ARM Neoverse-N1; 45 nm area/power overheads are 1.8%/2.2%.

Acknowledgments

This work was funded in part by NSF CAREER Award 2339317, NSF 2235398, and a UCR Dissertation Completion Fellowship. It was also partly funded by IITP grant RS-2025-02218733 (Full-Stack Development of a 1-POPS Digital Neuromorphic Event Processor) and IITP-ITRC grant IITP-2026-RS-2021-II211817, funded by the Korea government (MSIT).

References

[1] K. Al-Hawaj et al., "EVE: Ephemeral Vector Engines," *IEEE HPCA*, 2023.
[2] N. Binkert et al., "The gem5 Simulator," *ACM SIGARCH CAN*, 2011.
[3] C. Eckert et al., "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," *ACM/IEEE ISCA*, 2018.
[4] G. Gerganov, "llama.cpp: LLM Inference in C/C++," GitHub, 2024.
[5] S. G. Ghinani et al., "Enabling Low-Cost Secure Computing on Untrusted In-Memory Architectures," *USENIX Security*, 2025.
[6] M. Hong et al., "FlexInt: A New Number Format for Robust Sub-8-Bit Neural Network Inference," *IEEE/ACM ICCAD*, 2024.
[7] M. Lenjani et al., "Fulcrum: A Simplified Control and Access Mechanism toward Flexible and Practical In-Situ Accelerators," *IEEE HPCA*, 2020.
[8] S. Oh et al., "Non-Uniform Step Size Quantization for Accurate Post-Training Quantization," *ECCV*, 2022.
[9] J. Wei et al., "T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge," *ACM EuroSys*, 2025.
[10] J. Zhang et al., "Inhale: Enabling High-Performance and Energy-Efficient In-SRAM Cryptographic Hash for IoT," *IEEE/ACM ICCAD*, 2022.
[11] J. Zhang et al., "Sealer: In-SRAM AES for High-Performance and Low-Overhead Memory Encryption," *ACM/IEEE ISLPED*, 2022.
[12] J. Zhang et al., "BP-NTT: Fast and Compact In-SRAM Number Theoretic Transform with Bit-Parallel Modular Multiplication," *ACM/IEEE DAC*, 2023.
[13] J. Zhang et al., "CryptoSRAM: Enabling High-Throughput Cryptography on MCUs via In-SRAM Computing," arXiv, 2025.
[14] J. Zhang et al., "A Near-Cache Architectural Framework for Cryptographic Computing," arXiv, 2025.
[15] J. Zhang et al., "Unlocking Energy-Efficient and High-Throughput Secure Data Communication in IoT with Memory-Centric Computing," *IEEE IPDPS*, 2025.
[16] J. Zhang et al., "Meta-PIM: An End-to-End Compiler for Configurable Processing-in-Memory," *IEEE CAL*, 2026.
[17] J. Zhang et al., "No One-Size-Fits-All: A Workload-Driven Characterization of Bit-Parallel vs. Bit-Serial Data Layouts for Processing-Using-Memory," *IEEE IPDPS*, 2026.