

Lupin: Spatial Resource Stealing with Outlier-First Encoding for Mixed-Precision LLM Acceleration

Taein Kim*, Sukhyun Han*, Seongwook Kim*, Gwangeun Byeon*, Jungmin Lee[†], Seokin Hong*

*Department of Electrical and Computer Engineering, [†]Department of Semiconductor and Display Engineering
Sungkyunkwan University, Suwon, Korea
{xovnd2010, kavin1010, su8939, kebyun, leejm518, seokin}@skku.edu

Abstract—LLM inference often exceeds on-chip memory capacity, causing frequent external memory access. Quantization reduces memory cost but loses accuracy due to outliers. Prior mixed-precision accelerators address this issue with encoding schemes, but often result in accuracy degradation for LLMs and pipeline stalls. We present *Lupin*, an algorithm-architecture co-design with *Outlier-First Encoding*, which stores outliers in high precision by reallocating less critical normal values. This preserves maximal outlier representation and enables stall-free execution with low-precision MAC units. Experiments show that *Lupin* maintains accuracy while achieving a 2.02× speedup.

Index Terms—Large language models, Quantization, Mixed-precision, Outlier

I. INTRODUCTION

As Transformer-based Large Language Models (LLMs) [1] grow, inference increasingly exceeds on-chip memory capacity [2], resulting in frequent external memory accesses, performance bottlenecks [3], [4], and underutilization of compute hardware [5]–[9]. Quantization [10]–[12] mitigates this by using low-bit weights and activations, but rare large-magnitude outliers degrade accuracy [13]–[15]. This issue is especially severe for activations in attention mechanisms, where activation values directly determine attention probabilities [15], [16]. Therefore, proper handling of activation outliers is critical for maintaining accuracy. To address this challenge, we propose *Lupin*, a mixed-precision inference framework and hardware designed to handle outliers while preserving overall accuracy efficiently.

II. MOTIVATION & BACKGROUND

Existing mixed-precision approaches attempt to balance accuracy and efficiency but suffer from inherent limitations. OliVe [17] applied **mixed data-type quantization** (Fig. 1a) but still encodes outliers in 4 bits, resulting in substantial loss

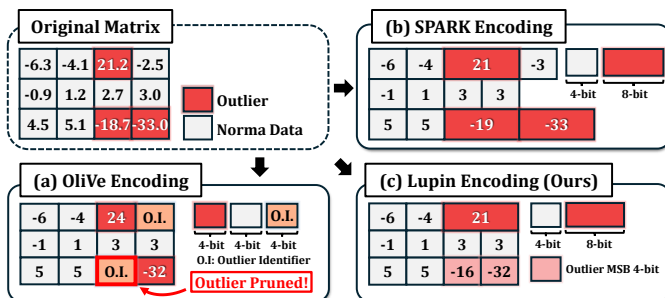


Fig. 1: Mixed-precision encoding comparison

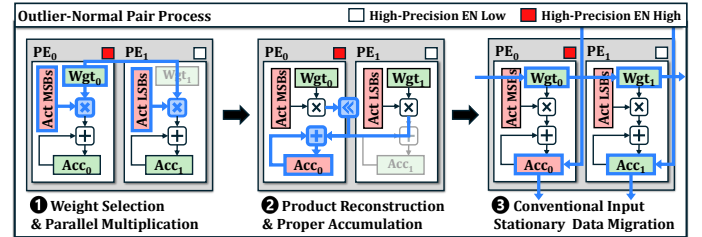


Fig. 2: Main dataflow for Lupin's stall-free computation: Outlier-Normal Pair processing

of outlier information. SPARK [18] applied **mixed bit-length quantization** (Fig. 1b), allocating 8-bit precision to outliers to improve accuracy, but its 4-bit-optimized compute units stall when handling higher-precision data, reducing throughput.

To overcome these limitations, *Lupin* introduces **Outlier-First Encoding** and a stall-free mixed-precision accelerator. Our encoding compress normal values in INT4, while outliers are preserved in INT8 by reallocating the storage space of the paired normal value.

III. OUTLIER-FIRST ENCODING

We introduce **Outlier-First Encoding** (Fig. 1c), a compact data encoding that efficiently handles outliers while ensuring compatibility with the hardware systems. This encoding is designed to work with a simple, fixed-size pair format (1-byte), making it suitable for integration with standard memory systems.

Normal-Normal Pair. Both values are quantized to low precision (e.g., INT4). This case achieves a high compression rate similar to conventional quantization schemes.

Outlier-Normal Pair. When an outlier is paired with a normal value, the normal value is pruned, and its storage space is reassigned. This allows the outlier to be stored with higher precision (e.g., INT8). The 1-byte pair format remains intact, avoiding disruption to a byte-based memory system.

Outlier-Outlier Pair. When two adjacent outliers occur, the most critical information from both is retained by storing the 4 MSBs of each outlier. This ensures that the encoding preserves key information even when dealing with multiple outliers.

As seen in Section V, our method preserves outlier information more effectively. Unlike OliVe, which represents outliers in a 4-bit float-based data type, our approach allows outliers to be represented with up to 8-bit precision. For an outlier pair, OliVe aggressively prunes one of the outliers, whereas

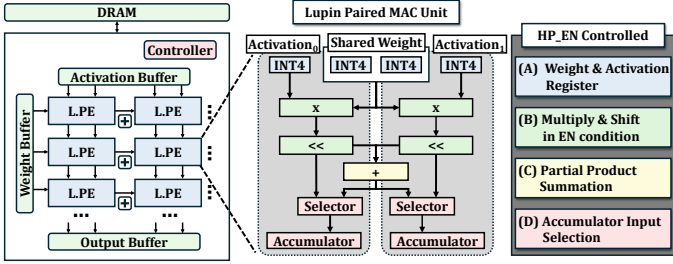


Fig. 3: Lupin Architecture. L.PE means Lupin PE in short

our method preserves the magnitudes of both values, retaining critical information and achieving higher accuracy.

Outlier indices are extracted during the encoding process and stored separately using the Block Sparse Index (BSI) [19]. For each data block loaded into the PE-array, BSI stores the relative distances of outliers, minimizing metadata size.

IV. LUPIN ARCHITECTURE

The Lupin Accelerator is designed to efficiently support mixed-precision computation without pipeline stalls caused by high-precision operations. It employs an input-stationary systolic array with paired 4-bit MAC units, where most operations are performed using INT4. **High-Precision Enable** (HP_EN) signal generated by outlier indices allows each Processing Element (PE) to switch to high-precision execution.

Fig. 2 illustrates how HP_EN enables high-precision operations to be executed on low-precision MAC units. Outlier indices decoding and HP_EN delivery are performed in parallel with activation block loading, aligning outlier control with the input stationary data flow. Guided by the HP_EN signal, each PE dynamically selects the appropriate weights (Fig. 2 ①) and accumulates partial products at higher precision.

The computation dataflow is divided into three cases, according to Outlier-First Encoding:

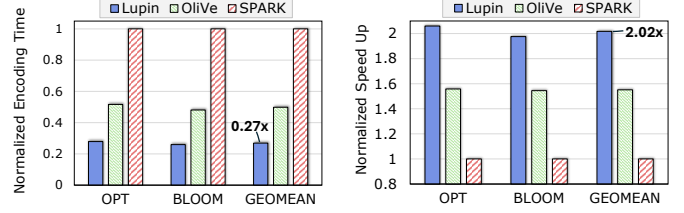
Normal-Normal Pair. This pair follows standard INT4 operations, processed as typical low-precision computations with independently operating MAC units.

Outlier-Outlier Pair. The dataflow for this pair is similar to the normal case, but the 4 MSBs are shifted to restore its magnitude and accumulated.

Outlier-Normal Pair. This operation constitutes the core mechanism of stall-free mixed-precision computation. When a normal value is pruned, the freed MAC unit is reassigned to process the high-precision outlier. Two 4-bit MAC units operate in parallel to simultaneously produce the MSB and LSB partial products of the outlier. The MSB result is shifted and combined

TABLE I: Perplexity comparison of LLM models on WikiText2 and C4 datasets (lower is better).

Model	OPT 1.3B		OPT 2.7B		BLOOM 1B7		BLOOM 3B	
	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4	WikiText2 / C4
FP16	14.62 / 14.72	12.47 / 13.16	15.39 / 17.97	13.48 / 16.14				
OliVe	46.93 / 45.87	36.38 / 36.60	18.82 / 21.51	16.55 / 19.68				
SPARK	36.86 / 37.88	27.52 / 29.33	21.16 / 23.92	17.45 / 21.31				
Lupin	16.78 / 16.99	13.78 / 14.71	17.69 / 20.53	14.87 / 17.90				



(a) Normalized data encoding time (b) Normalized overall speedup

Fig. 4: Runtime Encoding overhead and overall speedup comparison of different mixed-precision accelerator designs

with the LSB result (Fig. 2 ②), allowing the INT4 x INT8 multiplication to be completed within a single cycle.

Overall architecture, as shown in Fig. 3, supports this dataflow to enable stall-free execution with minimal control overhead and high hardware utilization.

V. EVALUATION

LLM Results. We evaluate Lupin on OPT [20] and BLOOM [21] models using post-training quantization [22], with WikiText2 [23] and C4 [24] datasets as shown in Table 1. Compared to baselines, OliVe [17] and SPARK [18], Lupin maintained near-FP16 accuracy and outperformed both baselines.

Area & Power. Hardware synthesis [25] in 28nm CMOS showed that Lupin reduced area to about 73% of both baselines and consumed only 76% of SPARK and 94% of OliVe core power, demonstrating its efficiency in hardware resource usage.

Runtime Quantization. In terms of quantization encoding overhead (Fig. 4a), an experiment was conducted on a Qualcomm Snapdragon 8 Gen 2 [26] with a C-based simulator. The proposed method achieved the lowest latency, requiring only about 54% of OliVe’s encoding time and 27% of SPARK’s. This advantage stems from its regular data pair packaging scheme. In contrast, OliVe incurs conditional-branching overhead for multi-type data handling, whereas SPARK incurs data rearrangement overhead to meet memory alignment requirements.

Performance. We evaluate compute latency using a cycle-level simulator based on DnnWeaver [27] and BitFusion [11]. Lupin and OliVe exhibit stall-free computation, whereas SPARK incurs stalls on high-precision data. As shown in Fig. 4b, Lupin achieves higher overall speedup due to shorter encoding time and the absence of pipeline stalls, resulting in 2.02x and 1.30x speedup over SPARK and OliVe, respectively.

VI. ACKNOWLEDGMENT

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.10692981, 50%), the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) (RS-2021-II212052, 25%) grant funded by the Korea government (Ministry of Science and ICT), and Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.00228970, 25%). Seokin Hong is the corresponding author.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, E. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [2] Z. Wang, Z. Chu, T. V. Doan, S. Ni, M. Yang, and W. Zhang, "History, development, and principles of large language models: an introductory survey," *AI and Ethics*, vol. 5, no. 3, pp. 1955–1971, 2025.
- [3] S. Yun, S. Park, H. Nam, Y. Lee, G. Lee, K. Kyung, S. Kim, N. S. Kim, J. Kim, H. Kim *et al.*, "The new llm bottleneck: A systems perspective on latent attention and mixture-of-experts," *arXiv preprint arXiv:2507.15465*, 2025.
- [4] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1080–1096.
- [5] S. Kim, G. Byeon, S. Kim, H. Kim, and S. Hong, "Conveyor: Towards asynchronous dataflow in systolic array to exploit unstructured sparsity," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 423–431.
- [6] S. Han, S. Kim, G. Byeon, J. Yoon, and S. Hong, "Zebra: Leveraging diagonal attention pattern for vision transformer accelerator," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [7] G. Byeon, S. Kim, H. Kim, S. Han, J. Kim, P. Nair, T. Kang, and S. Hong, "Avalanche: Optimizing cache utilization via matrix reordering for sparse matrix multiplication accelerator," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1746–1759.
- [8] Y. Jang, H. Cho, Y. Ryu, J. Kim, and S. Hong, "Pimpal: Accelerating llm inference on edge devices via in-dram arithmetic lookup," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.
- [9] H. Cho, Y. Jang, H. Kim, S. Kim, K. Kwon, G. Kim, and S. Hong, "Librapim: Dynamic load rebalancing to maximize utilization in pim-assisted llm inference systems," in *2025 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2025, pp. 43–56.
- [10] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.
- [11] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.
- [12] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.
- [13] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.
- [14] S. Son, W. Park, W. Han, K. Kim, and J. Lee, "Prefixing attention sinks can mitigate activation outliers for large language model quantization," *arXiv preprint arXiv:2406.12016*, 2024.
- [15] J. Park, T. Lee, C. Yoon, H. Hwang, and J. Kang, "Outlier-safe pre-training for robust 4-bit quantization of large language models," *arXiv preprint arXiv:2506.19697*, 2025.
- [16] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang, "Systematic outliers in large language models," *arXiv preprint arXiv:2502.06415*, 2025.
- [17] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.
- [18] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1029–1042.
- [19] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, K. Gopalakrishnan, and L. Chang, "Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.
- [20] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.
- [21] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," 2023.
- [22] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.
- [23] M. Stephen, X. Caiming, B. James, and R. Socher, "wikitext," 2016.
- [24] J. Dodge, A. Marasovic, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner, "Documenting large webtext corpora: A case study on the colossal clean crawled corpus," pp. 1286–1305, 2021.
- [25] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 1997.
- [26] Qualcomm Technologies, Inc. (2022, Nov.) Snapdragon 8 gen 2 mobile platform. Code name: Kalama (arm64). [Online]. Available: <https://www.qualcomm.com/products/snapdragon-8-gen-2-mobile-platform>
- [27] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration," in *the Workshop on Cognitive Architectures*, 2016.