

# Dolphium: Co-Optimizing Quantization Dataflow and Paradigms on Poly-Hierarchical NPUs

Xiuping Cui  
Peking University  
Beijing, China  
cuixiuping@pku.edu.cn

Chengrui Zhang  
Peking University  
Beijing, China  
zhangchr@stu.pku.edu.cn

Xiang Chen<sup>†</sup>  
Peking University  
Beijing, China  
xiang.chen@pku.edu.cn

Yun (Eric) Liang<sup>†</sup>  
Peking University  
Beijing, China  
ericlyun@pku.edu.cn

**Abstract**—Poly-hierarchical NPUs integrate distributed memory modules with heterogeneous computation units, posing significant challenges for mapping quantized operators. The difficulty arises from the need to coordinate data transfers across memory hierarchies and to assign diverse operations to suitable computation units. In this work, we systematically construct the mapping space from quantization to dataflows by addressing three key aspects: generation of NPU-friendly computation flows, integrated operation–data co-mapping, and determination of transfer granularity and frequency. Building on this foundation, we further exploit quantization dataflows to guide the selection of quantization paradigms. Compared with the state-of-the-art quantization compiler, our mapping achieves a 1.67–2.03× speedup. Moreover, the selected quantization paradigms deliver an average 2.18× efficiency improvement on NPUs without accuracy loss.

## I. INTRODUCTION

Neural processing units (NPUs) [1]–[4] have become a critical platform for accelerating neural networks, delivering high throughput through specialized tensor computation units and large on-chip memory resources. Modern NPUs, such as Ascend910 [2], TPU [3], and Simba [1], integrate numerous distributed memory modules and heterogeneous computation units, with each computation unit tightly coupled to its local memory block. Unlike hierarchical NPUs, which assume that all data follow a single path where they are loaded into registers, fully processed, and then written back to off-chip memory, modern NPUs require data to traverse multiple on-chip memory modules along different paths so that each data can be accessed by appropriate computation units. We refer to NPUs with such characteristics as *poly-hierarchical NPUs*. In such architectures, computational efficiency is determined not only by arithmetic intensity but also by how computation flows of the operators interact with the computation units and memory modules [5].

Quantization is a key technique with applications in neural network compression [6] and efficient deployment on hardware [7]–[10]. However, the characteristic of poly-hierarchical NPUs poses significant challenges for quantized operators [9]–[18] mapping. Quantized operators perform computations using low-precision data but require dequantization operations to restore data precision and maintain accuracy. Consequently, their computation flows are complex, involving multiple types of operations such as broadcasting, dequantization, and reduction. Moreover, a variety of quantization paradigms with differences

exists, and different paradigms lead to distinct computation flows with varying implications for efficiency. Without effective mapping strategies, the performance of quantized operators on poly-hierarchical NPUs can be severely degraded, either due to under-utilization of heterogeneous compute resources or excessive data transfers across memory modules.

Mapping quantized operators onto NPUs requires transforming computation flows into dataflows on hardware, specifically the transfers of data across memory modules. The complexity arises because quantized operators involve multiple operations, while NPUs incorporate numerous computation units and memory modules. These characteristics give rise to three fundamental challenges: (1) Transforming quantized operators into NPU-friendly computation flows in order to fully exploit heterogeneous compute resources. (2) Determining the assignment of operations to computation units and the placement of operands in memory modules in order to maximize computational parallelism and data locality. (3) Designing the granularity and frequency of data transfers to make effective use of limited on-chip memory resources and bandwidth.

Existing works [9], [10], [19], [20] have not fully addressed these challenges. Manually optimized operator libraries, such as Cann-ops-adv [19] and Atom [10], rely on handcrafted designs, limiting their applicability to specific operators on specific hardware. In contrast, general-purpose quantization compilers [20] neither explore the space of computation flows, nor construct systematic mappings of operations and data to hardware units, nor consider operator-specific characteristics when determining data transfers among memory modules. As a result, these methods provide limited support for efficiently mapping quantized operators onto poly-hierarchical NPUs.

In this work, we propose *Dolphium*, which addresses the three challenges and constructs a mapping space for quantized operators on poly-hierarchical NPUs, ultimately leveraging the quantization dataflows to guide the selection of quantization paradigms. (1) For *Challenge 1*, we adopt a two-stage approach to generate NPU-friendly computation flows. First, we apply multiple hardware-agnostic transformations to generate all feasible quantization flows. Then, we select the most efficient implementations during exploration, enabling generalization across different architectures. (2) For *Challenge 2*, we implement an integrated operation–data co-mapping strategy, assigning operations and data to computation units and memory

<sup>†</sup> Corresponding author.

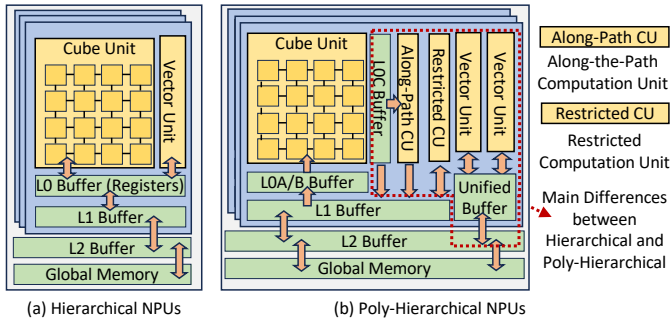


Fig. 1. (a) Hierarchical NPUs and (b) Poly-Hierarchical NPUs

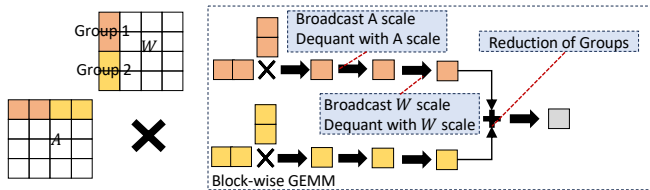


Fig. 2. An Example of Quantization Computation Flow

modules. By considering the computation flow as a whole rather than mapping each operation independently, we avoid under-utilization of computation units and excessive data transfers. (3) For *Challenge 3*, we determine the granularity and frequency of data transfers among memory modules using a data transfer pattern designed according to the computational characteristics of quantization. We leverage a performance model of poly-hierarchical NPUs to explore the above mapping space and identify quantization dataflows with minimal latency.

Finally, we use the dataflows to guide quantization paradigm selection at operator level, optimizing computational efficiency on NPUs while preserving accuracy.

Our contributions are summarized as follows:

- We present *Dolphium*, a framework that co-optimizes quantized operator mapping and quantization paradigm selection on poly-hierarchical NPUs.
- We construct the mapping space of quantized operators by addressing three key dimensions: NPU-friendly computation flow generation, integrated operation–memory co-mapping, and the determination of data transfer granularity and frequency.
- We exploit quantization dataflows to guide operator-level quantization paradigm selection, improving efficiency on NPUs while maintaining accuracy.

Our quantization mapping achieves a  $1.67\times$  to  $2.03\times$  performance improvement over state-of-the-art quantization compiler [20]. Furthermore, compared to state-of-the-art quantization paradigms [10], our paradigm selection attains a  $2.18\times$  speedup in average without accuracy loss.

## II. PRELIMINARY

### A. NPUs with Poly-Hierarchical Memory

A poly-hierarchical NPU comprises distributed memory modules and heterogeneous computation units tightly coupled to these modules, as shown in Fig. 1(b). The red box highlights

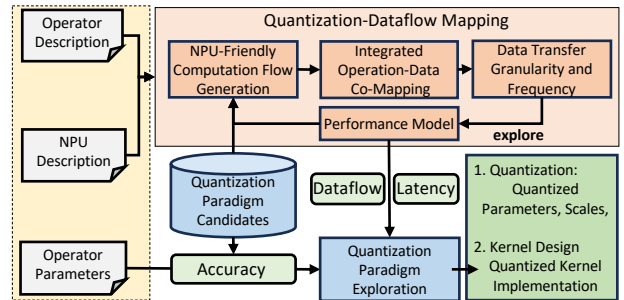


Fig. 3. Dolphium Overview

the distinctions from hierarchical NPUs: multiple computation units, additional memory modules, and more data paths. For instance, along-the-path units process data from LOC, restricted units operate on L1 buffer with limited operation supports, and vector units access data only from the unified buffer.

Dataflow design on hierarchical NPUs [21]–[28] is relatively simple: each operation is mapped to the computation unit with minimal latency, and all data traverse a single path from off-chip memory to registers and back. In contrast, poly-hierarchical NPUs necessitate explicit selection of both computation units and data paths.

### B. Quantization Paradigm and Computation Flow

A quantization paradigm is characterized by two parameters: the data bit-width and the block size. All elements within a block share a common quantization parameter, such as the scale factors. In block-wise quantization, the non-reduction dimension is often divided into groups of size 1. For example, in the  $(W_4G_{128}, A_4G_{128})$  paradigm, each weight element is quantized to 4 bits, and every 128 elements along the reduction dimension form a group with a shared scale factor.

The computation of a quantized operator involves multiple operations, including block-wise tensor operations, broadcasting, dequantization, and reduction. The ordered sequence of these operations is referred to as the *computation flow*, as illustrated in Fig. 2. Block-wise tensor operations are performed at the group level, broadcasting replicates scale factors along specific dimensions, dequantization restores data precision, and reduction aggregates outputs across groups.

Prior work [9], [10], [19] constructs computation flows to minimize computational cost by performing broadcasting and dequantization first on operands with a smaller group size. However, such flows are suboptimal for poly-hierarchical NPUs for two reasons. First, computation units support only limited operation types, so the default flow may underutilize hardware resources. Second, the default flow does not optimize on-chip data movement, potentially incurring excessive inter-memory module transfers.

## III. OVERVIEW

Fig. 3 presents an overview of *Dolphium*. Its inputs include operator descriptions, parameters, and NPU specifications. Leveraging these inputs, *Dolphium* maps quantized operators onto NPUs through three stages: (i) generating NPU-friendly computation flows, (ii) performing integrated operation–data

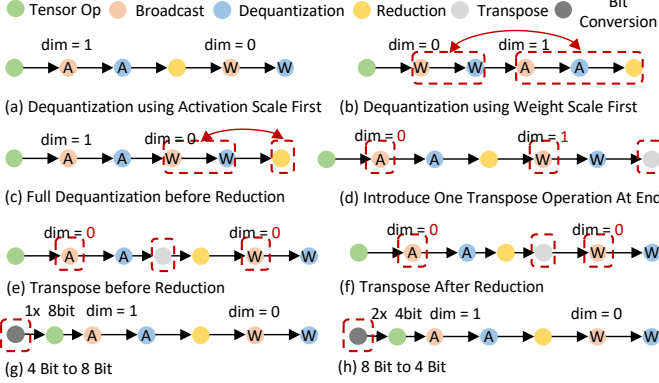


Fig. 4. Eight Different Computation Flows. (a)–(f) for Quantization Paradigm:  $(W_4, A_4G_{128})$ ; (g)–(h) for Paradigm  $(W_4, A_8G_{128})$ ; Red Arrows and Blocks in (b)–(f) Indicates Their Differences from (a).

co-mapping onto on-chip computation units and memory modules, and (iii) determining the granularity and frequency of data transfers. Using a performance model tailored for poly-hierarchical NPUs, Dolphium explores the mapping space to identify high-efficiency quantization dataflows. These dataflows further guide quantization paradigm selection, optimizing network latency while maintaining accuracy. Finally, Dolphium outputs both quantized operator parameters and kernel implementations, providing a comprehensive solution for efficient deployment of neural networks on NPUs.

#### IV. QUANTIZATION-DATAFLOW MAPPING ON POLY-HIERARCHICAL NPU

This section presents the mapping stages from quantized operators to dataflows on NPUs, and then introduce a performance model on poly-hierarchical NPUs to explore the mapping space.

##### A. NPU-Friendly Quant. Computation Flow Generation

The computation flow of a quantized operator critically affects efficiency, as different flows can lead to large variations in compute utilization and data movement. Prior work [9], [10], [19], [20] generally prioritizes flows with minimal computational cost; however, cost alone does not guarantee efficiency. Low-cost flows may still suffer from poor hardware utilization or excessive memory transfers. To address this limitation, we develop a systematic flow generation strategy that explicitly targets NPU-friendly execution. Our method adopts a two-stage design: hardware-agnostic flow generation followed by hardware-aware flow selection. The generated candidates include efficient flows, from which the optimal one is later chosen. This subsection focuses on the flow generation stage.

The efficiency of quantization computation flows is determined by two factors: the types of operations and their ordering. Different types achieve different execution efficiencies on specific computation units and may even constrain mapping choices. For example, broadcasting along the first dimension can be mapped onto along-the-path CUs, whereas broadcasting along the second dimension cannot. Meanwhile, operation ordering shapes data dependencies within the flow and, consequently, influences the volume of data transfer.

To generate a diverse set of computation flows, we apply two categories of transformations. (1) *Operation reordering*, which changes the execution sequence, such as adjusting the placement of dequantization or swapping dequantization with reduction, as illustrated in Fig. 4(a)–(c). (2) *Operation augmentation*, which alters the types of operations by introducing additional transformations such as transpose and bit-width conversion. A transpose converts  $(A \times W)$  into  $((W^T \times A^T)^T)$ , as shown in Fig. 4(d)–(f), thereby modifying the broadcasting dimension. Bit-width conversion reconciles operands with different precisions: low-precision data may be promoted, while high-precision data may be split into multiple low-precision values, as illustrated in Fig. 4(g) and (h) for 4-to-8-bit promotion and 8-to-4-bit splitting, respectively.

##### B. Integrated Operation–Data Co-Mapping

Mapping operations to computation units and data to memory modules is crucial, as it determines workload distribution across heterogeneous resources and the volume of data transfers. Conventional quantization compilers [20] typically assign each operation to the computation unit with minimal latency. However, this approach often underutilizes resources and overlooks data movement, leading to suboptimal end-to-end performance. To address this, we adopt an integrated operation–data co-mapping strategy that jointly considers all operations and their associated data, optimizing both computation and communication globally.

To implement integrated mapping, we first serialize memory modules and computation units into a single linear sequence. Since only block-wise tensor operations can execute on cube units, the sequence contains cube units exactly once, forming a path from vector units to cube units and back. Fig. 5(b) is the serialization of Fig. 1(b), where yellow blocks denote computation units and blue blocks denote memory modules.

Each computation flow is then sequentially mapped to the unit sequence by assigning operations to supported computation units, ensuring that each operation is mapped to a unit no earlier than those assigned to preceding operations. Examples are shown in Fig. 5(c)–(f). Notably, (e) and (f) correspond to the same quantization paradigm but differ due to a transpose: in (e), the broadcast operation cannot use the along-the-path CU and remains on vector units, whereas in (f), the transpose modifies the broadcast dimension, enabling mapping to the along-the-path CU and reducing vector-unit load at the cost of an additional final transpose. Sequential mapping does not require assigning each operation to the nearest unit; lower-performance units may be skipped to avoid bottlenecks.

##### C. Granularity and Frequency of Multi-Path Data Transfers

Mapping data to memory modules requires determining both the tile size at each memory level (i.e., transfer granularity) and the frequency of inter-module transfers, since a single module cannot store an entire tensor. This task is challenging due to heterogeneous access patterns across operations and varying memory capacities. Existing approaches [19], [20] that directly reuse transfer strategies for non-quantized operators are inefficient, as they neglect the unique characteristics of quantized operators, resulting in unnecessary data movement.

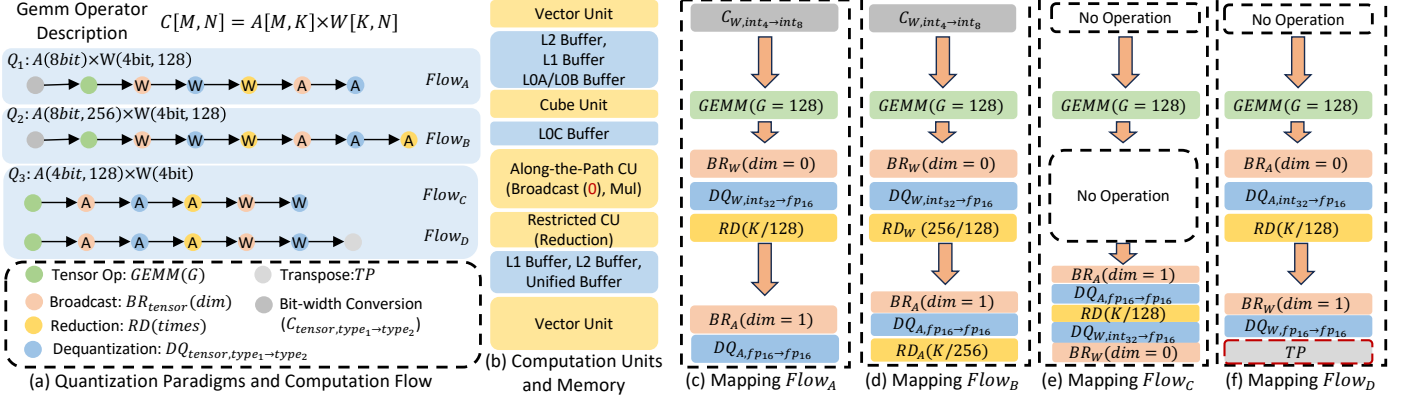


Fig. 5. Four examples of data transfer construction. (a) Three quantization paradigms and four corresponding computation flows; (b) The serialized computation units and memory modules, where yellow denotes computation units and blue indicates the memory modules involved; (c)–(f) Examples of operation-data mapping, where different colors represent different operation types and arrows indicate the memory modules traversed during data movement.

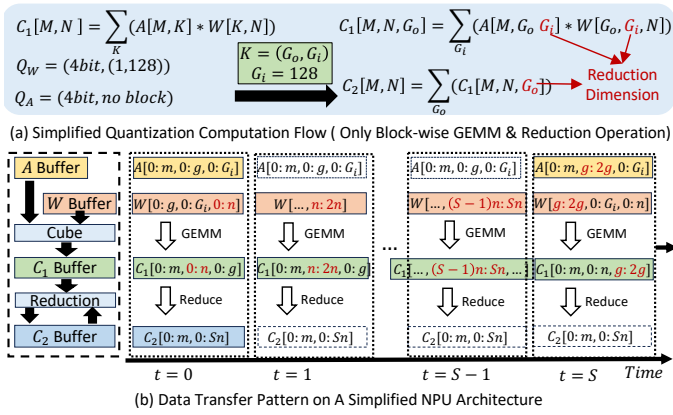


Fig. 6. Illustration of Data Transfer for a Simplified Computation Flow on a Simplified NPU Architecture. The notation  $0:g$  in the block coordinates indicates that the dimension ranges from 0 to  $g$ , while the red highlight denotes the dimension that changes between adjacent blocks.

To address this, we design a transfer pattern that adapts both granularity and frequency to these characteristics.

Fig. 6(a) illustrates a simplified computation flow of a quantized operator, focusing on block-wise tensor & computations and reductions, which offer high data reuse potential. For block-wise computations, low-precision inputs but high-precision outputs ( $C_1$ , int32) make minimizing output transfers crucial. Additionally, reduced dimensions (from thousands to hundreds) and smaller input bit-widths allow multiple input groups to be stored and processed per pass. Reduction operations repeatedly reuse outputs, so keeping them in on-chip memory avoids redundant transfers.

This motivates a *hybrid input–output stationary* pattern: block-wise tensor computations are both input- and output-stationary, while reduction operations follow an output-stationary scheme to maximize on-chip reuse.

Fig. 6(b) illustrates a simplified hardware example and the corresponding temporal data transfers, omitting broadcast and dequantization for clarity. The  $A$  buffer holds a block of size  $m \times g \times G_i$  and advances after processing  $S$  incoming  $W$  blocks.  $W$  streams along its last dimension,  $C_1$  streams along the second dimension to the reduction unit, and  $C_2$  stores a block of size  $m \times Sn$  as a resident tile. Consequently,  $A$  follows a block-wise input–output stationary pattern,  $C_1$  transfers occur

only after each block is computed (output-stationary), and  $C_2$  writes back after computation completes (also output-stationary). Block-wise input-stationary is used for  $A$  rather than full stationary because  $C_2$ 's limited capacity constrains the number of incoming blocks  $S$ . This transfer pattern generalizes to more complex architectures, with parameters  $\{m, n, g, S\}$  determined by on-chip buffer sizes.

#### D. Performance Model for Poly-Hierarchical Architecture

Existing dataflow models [21], [22] assume hierarchical NPUs with a single data path, and thus cannot guide mapping on poly-hierarchical architectures. We propose a model that accounts for multiple computation units and memory modules to enable accurate latency estimation.

The performance model is constructed by individually evaluating the latency contributions of all memory and computation units. For a memory module, its latency is defined as the total amount of data transferred through it divided by its bandwidth. For a computation unit, the latency is modeled as the number of invocations divided by its operating frequency.

Different types of resources contribute to the overall execution time in distinct ways. We classify resources into *shared* and *private*. Shared resources (e.g., L2 buffer) are accessible across cores, while private resources (e.g., L1 buffer, Cube Unit) are used within a single core. The total latency is given by:

$$lat = \max \left( \max_{u \in \text{shared}} \left( \sum_{\text{core}_i} lat_{u, \text{core}_i} \right), \max_{u \in \text{private}} (lat_u) \right). \quad (1)$$

On right-hand side of Eq. 1, the first term sums the latency of each shared resource across all cores, and then takes the maximum over all shared resources to obtain the total latency of shared resources. The second term takes the maximum latency among the private resources within each core to evaluate the execution time inside a core.

Based on this performance model, we perform a brute-force exploration of the mapping space defined by the three steps above, identifying the dataflow with the lowest latency.

## V. DOLPHIUM: QUANT. PARADIGM EXPLORATION

In this section, we exploit the quantization dataflows to guide the choice of quantization paradigms, aiming to optimize computational efficiency on NPUs without compromising accuracy.

### A. Operator-Level Quantization Paradigm Selection

Conventional quantization methods [9]–[11] typically employ a single paradigm across the entire network. This design has two drawbacks. First, the computational efficiency on poly-hierarchical NPUs is highly sensitive to the paradigm; optimizing solely for accuracy may lead to inefficient execution. Second, it neglects substantial variation across operator-specific data distributions, where a paradigm optimal for one operator may be suboptimal for another. These issues motivate the need for operator-level quantization paradigm selection.

We formulate operator-level quantization as

$$\text{Accuracy} \geq s, \quad \min \sum_i \text{Latency}(\text{op}_i, Q_i), \quad (2)$$

where  $s$  is the target accuracy, and the objective is to minimize total network latency under this constraint. The candidate quantization paradigms include four primary modes: Q1 ( $W_4G_x, A_4G_x$ ), Q2 ( $W_4G_x, A_8$ ), Q3 ( $W_4, A_4G_x$ ), and Q4 ( $W_8, A_8$ ), which can be further instantiated with different group sizes  $x$  to adapt to individual operators.

### B. Quantization Paradigm Exploration

Directly optimizing Eq. 2 is infeasible due to the enormous design space of  $O(N^{\{Q\}})$ , necessitating per-operator exploration of quantization paradigms. The key challenge is evaluating the impact of a candidate paradigm on an individual operator without executing the full network.

For each operator, we estimate quantization error using the mean squared error (MSE):

$$\text{error} = \|(A + \delta A)(W + \delta W) - AW\|_2,$$

where  $\delta$  denotes quantization-induced error, and  $A$  is drawn from a calibration set. Candidate paradigms must satisfy

$$\text{error} \leq \alpha \cdot \text{error}_{\text{baseline}},$$

with  $\alpha$  controlling the allowable error tolerance and the baseline defined by a paradigm that ensures network accuracy, e.g., ( $W_4G_{128}, A_4G_{128}$ ). Among all feasible paradigms, the one minimizing latency is selected. The hyperparameter  $\alpha$  is kept uniform across all operators in a network, initialized with a large value and progressively reduced until the quantized network achieves accuracy comparable to the baseline.

## VI. EVALUATION

### A. Experiment Setup

We implement a cycle-accurate NPU simulator to conduct our experiments on two platforms: NPU1 (Fig. 1(b)) and Ascend 910B [2], denoted as NPU2. Detailed specifications of both platforms are provided in Table I. A key architectural distinction is that in NPU2, outputs from L0C are directly committed to L2 after computation, bypassing L1. We adopt Ladder [20] as our baseline. Evaluation focuses on linear

TABLE I  
HARDWARE SPECIFICATION

Component	Specification
Cube Unit ( $M \times K \times N$ )	Int4(Int8): $16 \times 32(64) \times 16$
Vector Unit	size = 128 lanes
Along-the-Path CU	size = 64 lanes, Broadcast(first dimension), Multiplication, ...
Restricted CU	16 lanes, Reduction, Transpose, ...
L0A / L0B / L0C Buffer	64 KB / 64KB / 128KB
Unified / L2 Buffer	512 KB / 192MB
L1 Buffer	1MB(NPU1) or 512KB(NPU2)
Number of Cores / Frequency	16(NPU1) or 20(NPU2) / 1 GHz
GM/L2 Bandwidth	1.6 TB/s / 5 TB/s

TABLE II  
GEMM BENCHMARKS

$M$	$(N, K)$ pairs			
2048, 8192	(4096, 11008)	(11008, 4096)	(6656, 17920)	(17920, 6656)
	(8192, 22016)	(22016, 8192)	(4096, 14336)	(14336, 4096)

operators from three large language models, Llama2-7B [29], Llama1-30B [30], and Mixtral-7x22B [31], with details summarized in Table II.

### B. Computational Efficiency of Quantized Operators

Fig. 7 illustrates the computational efficiency of two quantization paradigms, Q1 and Q2, across two platforms. The y-axis denotes speedup relative to the baseline, and the x-axis corresponds to the GEMMs in Table II. The speedups range from  $1.67\times$  to  $2.03\times$ . Performance differences arise across both paradigms and platforms: on NPU1, Q1 and Q2 achieve average speedups of  $1.78\times$  and  $1.9\times$ , respectively, while Q2 reaches  $1.78\times$  on NPU2. These variations are driven by two factors. First, distinct quantization paradigms induce different computation flows, which offer varying optimization opportunities. Second, heterogeneous memory hierarchies across NPUs affect the efficiency of the same flow. For instance, NPU1 can perform reductions within the L1 buffer with minimal L2 accesses, amplifying the benefits of the hybrid input-output stationary pattern.

### C. Quantization Dataflow Analysis

Fig. 8–10 provide a detailed analysis of each stage of the mapping procedure under two quantization paradigms, Q2 and Q3. Each bar reports the geometric mean of speedups (or data transfer volume reductions) across 16 GEMM instances, with the x-axis denoting the group size of the paradigms.

1) *Computation Flow*: Fig. 8 shows that optimizing computation flows significantly improves efficiency, achieving  $1.01\times$ – $1.99\times$  gains on NPU1 and  $1.04\times$ – $1.85\times$  on NPU2. The improvement varies with the quantization paradigm, group size, and NPU architecture.

2) *Operation-Data Co-Mapping*: Fig. 9 illustrates the impact of operation–data co-mapping on computational efficiency. The baseline uses only cube and vector units. Our method achieves  $1.05$ – $1.99\times$  speedup on NPU1 and  $1.04$ – $1.27\times$  on NPU2. The improvements arise from (i) distributing operations across multiple computation units to relieve vector-unit pressure and (ii) performing reductions earlier to reduce data movement, especially between the L2 buffer and unified buffer.

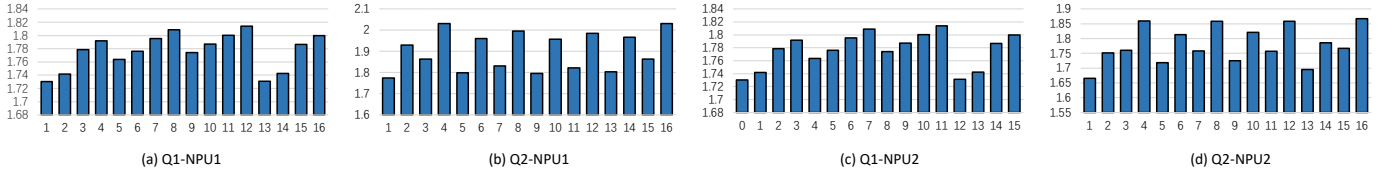


Fig. 7. Relative Latency of Quantization Paradigms Q1 ( $W_4G_{128}, A_4G_{128}$ ) and Q2 ( $W_4G_{128}, A_8$ ) on NPU1 and NPU2.

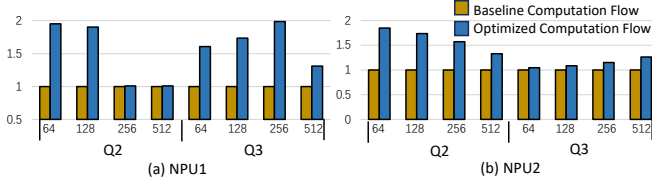


Fig. 8. Relative Latency of Different Quantization Computation Flows. Q2: ( $W_4G_x, A_8$ ); Q3: ( $W_4, A_4G_x$ )

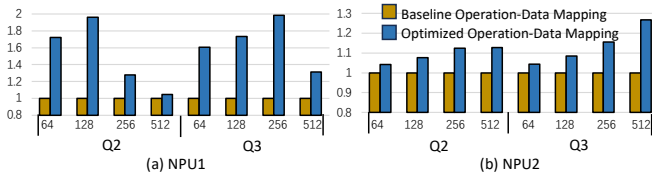


Fig. 9. Relative Latency of Different Operation and Data Mapping Strategies. Q2: ( $W_4G_x, A_8$ ); Q3: ( $W_4, A_4G_x$ )

3) *Transfer Granularity and Frequency*: Fig. 10 demonstrates the effectiveness of our optimized quantization transfer pattern in reducing data movement, focusing on the L2 buffer where the primary access bottleneck occurs. Compared with the output-stationary design in Ladder [20], our approach decreases data transfers by 44%–47% on NPU1 and 4.3%–30% on NPU2. This improvement is primarily attributed to two factors: (i) the hybrid input–output stationary pattern significantly reduces input data movement and (ii) retaining intermediate outputs on-chip minimizes data exchanges during reduction operations.

#### D. Computational Efficiency of Quantization Paradigms

Fig. 11 presents the computational efficiency of different quantization paradigms across NPUs, where the x-axis denotes group size and the y-axis denotes throughput. Throughput generally increases with larger group sizes but gradually plateaus as the workload on non-cube units decreases and data transfers are reduced. As a result, efficiency eventually converges to the computational capacity of cube units. Furthermore, quantization paradigms (Q1–Q4) exhibit notable differences in efficiency, with the extent of variation dependent on group sizes. The substantial differences in computational efficiency across quantization paradigms underscore the necessity of leveraging quantization dataflows to guide paradigm selection.

#### E. Effectiveness of Quantization Paradigm Exploration

We conduct quantization paradigm selection on NPU2, using Atom [10] as the baseline. Table III reports accuracy and speedup. Model quality is evaluated by perplexity (PPL) on the Wikitext-2 [32] dataset and by accuracy on three zero-shot benchmarks: PIQA [33], ARC-e [34], and ARC-c [34]. The runtime is measured only on the quantized linear operators as defined in [10], considering exclusively the prefill phase with input sequence lengths ranging from 1024 to 4096. The

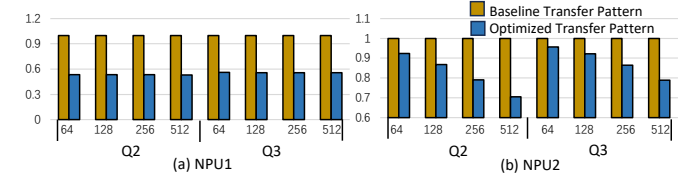


Fig. 10. Relative Data Transfer Volume of Different Data Transfer Patterns. Q2: ( $W_4G_x, A_8$ ); Q3: ( $W_4, A_4G_x$ )

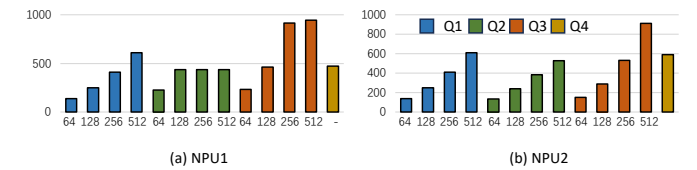


Fig. 11. Throughput of Different Quantization Paradigms on NPU1 and NPU2. Q1: ( $W_4G_x, A_4G_x$ ); Q2: ( $W_4G_x, A_8$ ); Q3: ( $W_4, A_4G_x$ ); Q4: ( $W_8, A_8$ )

results demonstrate that the proposed quantization paradigms deliver an average  $2.18\times$  speedup over the baseline without any accuracy degradation. These results highlight the importance of selecting different quantization paradigms for individual operators within a neural network. Operator-level selection not only significantly improves computational efficiency on NPUs, but also enables the choice of paradigms that better match the data distribution of each operator, yielding accuracy gains.

TABLE III  
ACCURACY AND SPEEDUP

Model	Q	PPL	Accuracy			Speedup
			PIQA	ARC-e	ARC-c	
Llama2-7B	Atom	6.12	75.14	52.99	38.40	2.09
	Ours	6.02	77.20	69.15	42.08	
Llama2-13B	Atom	5.31	76.50	57.49	42.32	2.28
	Ours	5.25	79.13	74.25	48.02	

## VII. CONCLUSION

In this work, we construct the mapping space of quantized operators on poly-hierarchical NPUs by addressing three key aspects: NPU-friendly computation flow generation, operation–data co-mapping, and determination of data transfer granularity and frequency. Building on this foundation, we further leverage quantization dataflows to guide the selection of quantization paradigms on NPUs. Compared with state-of-the-art approaches, our mapping achieves  $1.67\times$ – $2.03\times$  speedup. Moreover, on Ascend 910B, the selected quantization paradigms deliver an average of  $2.18\times$  higher efficiency than state-of-the-art paradigms, without accuracy degradation.

## ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation of China (Grant No. T2325001).

## REFERENCES

- [1] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucec Khailany, and Stephen W. Keckler. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *MICRO*, 2019.
- [2] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. In *HPCA*, 2021.
- [3] Norm Jouppi et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *ISCA*, 2023.
- [4] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Dianna: a small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGARCH Comput. Archit. News*, 2014.
- [5] Yuhang Zhou, Zhibin Wang, Guyue Liu, Shipeng Li, Xi Lin, Zibo Wang, Yongzhong Wang, Fuchun Wei, Jingyi Zhang, Zhiheng Hu, Yanlin Liu, Chunsheng Li, Ziyang Zhang, Yaoyuan Wang, Bin Zhou, Wanchun Dou, Guihai Chen, and Chen Tian. Squeezing operator performance potential for the ascend architecture. In *ASPLOS*, 2025.
- [6] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks, 2020.
- [7] Zhanhong Tan, Hongyu Cai, Runpei Dong, and Kaisheng Ma. Nnbaton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators. In *ISCA*, 2021.
- [8] Jingwei Cai, Yuchen Wei, Zuocong Wu, Sen Peng, and Kaisheng Ma. Inter-layer scheduling space definition and exploration for tiled accelerators. In *ISCA*, 2023.
- [9] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2025.
- [10] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In *MLSys*, 2024.
- [11] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *ICML*, 2023.
- [12] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*, 2024.
- [13] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- [14] Wenqi Shao et al. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.
- [15] Ying Zhang, Peng Zhang, Mincong Huang, Jingyang Xiang, Yujie Wang, Chao Wang, Yineng Zhang, Lei Yu, Chuan Liu, and Wei Lin. Qqq: Quality quattuor-bit quantization for large language models, 2024.
- [16] Renjie Wei, Songqiang Xu, Linfeng Zhong, Zebin Yang, Qingyu Guo, Yuan Wang, Runsheng Wang, and Meng Li. Lightmamba: Efficient mamba acceleration on fpga with quantization and hardware co-design. In *DATE*, 2025.
- [17] Wei Tao, Bin Zhang, Xiaoyang Qu, Jiguang Wan, and Jianzong Wang. Cocktail: Chunk-adaptive mixed-precision quantization for long-context llm inference. In *DATE*, 2025.
- [18] Zongwu Wang, Fangxin Liu, Peng Xu, Qingxiao Sun, Junping Zhao, and Li Jiang. Evasion: Efficient kv cache compression via product quantization. In *DATE*, 2025.
- [19] Huawei. Cann-ops-adv operator library, 2025.
- [20] Lei Wang et al. Ladder: Enabling efficient Low-Precision deep learning computing through hardware-aware tensor transformation. In *OSDI*, 2024.
- [21] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *MICRO*, 2019.
- [22] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucec Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *ISPASS*, 2019.
- [23] Size Zheng, Siyuan Chen, Siyuan Gao, Liancheng Jia, Guangyu Sun, Runsheng Wang, and Yun Liang. Tileflow: A framework for modeling fusion dataflow via tree-based analysis. In *MICRO*, 2023.
- [24] Size Zheng, Siyuan Chen, Peidi Song, Renze Chen, Xiuhong Li, Shengen Yan, Dahua Lin, Jingwen Leng, and Yun Liang. Chimera: An analytical optimizing framework for effective compute-intensive operators fusion. In *HPCA*, 2023.
- [25] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *ASPLOS*, 2020.
- [26] Size Zheng, Renze Chen, Anjiang Wei, Yicheng Jin, Qin Han, Liqiang Lu, Bingyang Wu, Xiuhong Li, Shengen Yan, and Yun Liang. Amos: enabling automatic mapping for tensor computations on spatial accelerators with hardware abstraction. In *ISCA*, 2022.
- [27] Renze Chen, Zijian Ding, Size Zheng, Meng Li, and Yun Liang. Motenn: Memory optimization via fine-grained scheduling for deep neural networks on tiny devices. In *DAC*, 2024.
- [28] Renze Chen, Zijian Ding, Size Zheng, Chengrui Zhang, Jingwen Leng, Xuanzhe Liu, and Yun Liang. Magis: Memory optimization via coordinated graph transformation and scheduling for dnn. In *ASPLOS*, 2024.
- [29] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shrutu Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
- [31] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L lio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Th ophile Gervet, Thibaut Lavril, Thomas Wang, Timoth e Lacroix, and William El Sayed. Mixtral of experts, 2024.
- [32] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.
- [33] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.
- [34] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.