Gundam: A Generalized Unified Design and Analysis Model for Matrix Multiplication on Edge

Quan Cheng[1,2], Haoyuan Li[1], Weirong Dong[1], Mingqiang Huang[2], Longyang Lin[2], Masanori Hashimoto[1*]

[1]Department of Informatics, Kyoto University, Kyoto, Japan
[2]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China
[*]{hashimoto@i.kyoto-u.ac.jp}

***Abstract*—Matrix multiplication is the core operation in many edge-AI applications, yet its efficient implementation requires balancing compute throughput with strict area and resource constraints. To address this, we propose Gundam, a generalized unified design and analysis model that enables agile and structured estimation and configuration of AI accelerator architectures. Gundam provides analytical modeling of matrix operations, supporting rapid evaluation of processing element size, data reuse pattern, buffer size, and computation latency under diverse hardware constraints. Unlike conventional models, Gundam jointly captures both hardware mapping and resource allocation within a unified model, facilitating fast and resource-aware design space exploration. To validate its accuracy and utility, we apply Gundam to guide accelerator generation across 16nm, 22nm, and 28nm process nodes. Results show that Gundam's estimated configurations differ by less than 6% from post-layout implementations, while automatically identifying optimal AI accelerator configurations under fixed resource constraints. Gundam offers a lightweight yet powerful tool for early-stage deployment and optimization of matrix processors on edge.**

***Index Terms*—Matrix multiplication, resource analysis, PE array modeling, AI accelerator configuration**

I. Introduction

Matrix multiplication lies at the main operation of modern artificial intelligence (AI) workloads, underpinning key operations in neural networks (NNs) such as fully connected layers, and attention mechanisms. Due to its regular structure and high computational density, it is often the primary target for hardware acceleration in AI systems [1], [2]. However, matrix multiplication also constitutes one of the most resource-intensive operations in AI inference and training, accounting for the majority of computational cost and energy consumption [3], [4]. This challenge becomes especially critical in edge-AI platforms, where resource budgets are significantly constrained in terms of area, power, and memory bandwidth. Efficiently deploying AI workloads on such platforms requires not only fast computation, but also careful planning of data movement, memory usage, and process element (PE) size.

Despite the extensive research on matrix engines and AI accelerators, many existing designs focus predominantly on maximizing raw compute throughput (e.g., peak PE utilization or array-level energy efficiency) while often neglecting the actual mapping behavior of AI models and the deployment constraints specific to edge scenarios [5]. In particular, challenges such as suboptimal data flow scheduling, inefficient use of chips' resources, and inflexible logic strategies are often overlooked in the early stages of the design [6]–[9].

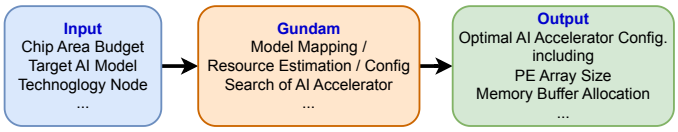

Fig. 1. Gundam: Conceptual Flow for Agile AI Accelerator Deployment.

To address these challenges, we propose Gundam: a **G**eneralized **Un**ified **D**esign and **A**nalysis **M**odel. Gundam presents a structural design and analysis model that systematically supports the decomposition and mapping of matrix multiplication for AI accelerators. This unified model (Fig. 1) bridges the gap between high-level AI model requirements and low-level hardware implementation constraints, offering a lightweight and scalable design methodology. By capturing both the spatial and temporal structure of matrix execution, Gundam supports fast design-space exploration and enables hardware-aware mapping and configuration tailored to edge-AI deployment. The main contributions are as follows:

- **Matrix Decomposition Modeling and Hardware Mapping:** A structured analysis is introduced to transform matrix dimensions into analytically tractable patterns of computation and data movement. This representation enables systematic decomposition strategies that enhance data reuse, computation parallelism, and architectural efficiency, while also reducing the need for data reshaping between adjacent layers.
- **Rapid Resource Estimation and Allocation for AI Accelerator:** An efficient search algorithm is delveloped for rapid estimation of optimal AI accelerator configurations (e.g., PE array size, memory size, and buffer partitioning) under chip area and design constraints. Gundam enables fast exploration across different NN models and process nodes (16/22/28nm), achieving less than 6% deviation from post-layout hardware results and matching the performance of state-of-the-art (SOTA) designs.

II. Related Work

Recent years have witnessed a surge in research on optimizing matrix multiplication for edge-AI deployment, with a particular focus on hardware-aware resource allocation and rapid, adaptable configuration under strict constraints. Early

work such as ViTA [10] introduced a configurable accelerator for Vision Transformer inference, leveraging pipelined heads and MLP optimizations to achieve high PE utilization and sub-watt power on edge platforms. However, while ViTA delivers impressive efficiency for specific workloads, it lacks a unified analytical model capable of generalizing to arbitrary matrix dimensions or technology nodes.

Other efforts, such as EdgeLLM [11], present CPU–FPGA heterogeneous architectures targeting LLM inference with custom compute engines and structured sparsity. These approaches automate compilation and support high energy efficiency, but are often tied to FPGA-specific features and do not offer a universal methodology for early-stage ASIC resource estimation or rapid design space exploration.

Analytical modeling tools such as MAESTRO [12] and TIMELoop [13] formalize dataflow and tiling strategies, providing reuse-aware analysis for AI accelerators. While they are effective at analyzing and mapping conventional dataflow patterns, these tools do not address the problem of optimal AI configuration search under explicit resource constraints. More recent efforts, such as Stream-HLS [14], propose automation frameworks with global optimization and streaming capabilities, but primarily target FPGA HLS and focus on dataflow scheduling rather than early-stage architectural resource planning. Therefore, the applicability of these approaches is limited when holistic resource allocation and hardware-aware design-space exploration are required for efficient edge-AI deployment. On the other hand, broader surveys have addressed system-level optimization on edge, including data partitioning and computation offloading strategies [15], [16], but tend to focus on generic trends rather than providing hardware-level resource allocation and analysis for AI accelerators.

Therefore, existing solutions are often constrained by device-specific assumptions, static dataflow models, or an inability to provide fast and accurate resource estimation across different technology nodes and workload requirements. To address these limitations, our work, Gundam, introduces a unified modeling framework capable of rapid and precise analysis of PE allocation, memory partitioning, and computation cycles for matrix multiplication accelerators under realistic resource constraints and diverse deployment scenarios.

III. Gundam Model

Modern AI workloads often exhibit highly repetitive and cyclic memory access patterns, particularly in matrix multiplication and convolution operations. Frequently, the same input data or weights are reused across multiple computation cycles. After each round of multiply-accumulate (MAC) operations, memory access typically returns to the initial address range, resulting in a looped or periodic dataflow. This regularity underpins the potential for efficient data reuse and resource sharing in edge-AI accelerators. By modeling these patterns, Gundam enables analytical estimation of processing efficiency, buffer utilization, and computational throughput, providing a foundation for systematic optimization of memory allocation and parallel computation in resource-constrained designs.

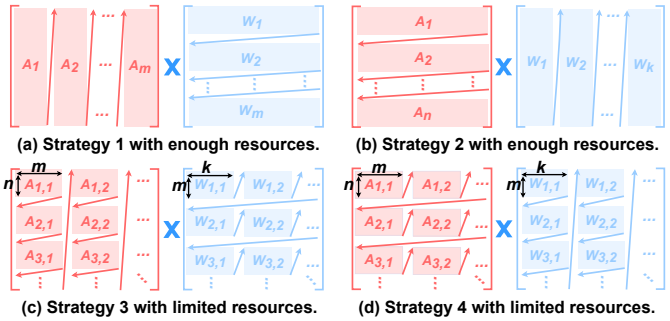

Fig. 2. Different Strategies of Matrix Multiplication Decomposition.

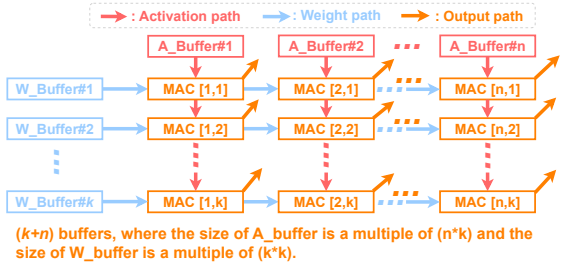

Fig. 3. Example of Mapping Between Matrix Multiplication and Hardware.

A. Matrix Multiplication Decomposition Analysis

Fig. 2 summarizes four representative matrix multiplication decomposition strategies. Strategies (a) and (b) directly map the computation to either row- or column-wise data access, which eliminates the need for intermediate result storage and aligns well with MAC-centric pipelines. However, for large-scale models on edge devices, these approaches are rarely feasible, as storing the entire activation and weight matrices on-chip quickly exceeds memory constraints. Strategies (a) and (c) also have a major drawback: their decomposition style generates a substantial amount of intermediate results, requiring significant on-chip temporary storage and introducing additional data movement. This is generally inefficient and deviates from typical matrix computations.

Strategy (d), by contrast, provides the most practical decomposition for edge accelerators. Here, the large matrix multiplication is split into multiple independent $n \times m$ and $m \times k$ submatrix blocks, allowing fine-grained scheduling and maximizing local data reuse. However, in multi-layer NN models, where the output of one layer directly feeds into the next, it is highly desirable to maintain a consistent activation format between layers. If the decomposition produces $n \times m$ input blocks and $n \times k$ output blocks, the mismatch in block dimensions introduces additional data reshaping and transfer overhead. Thus, to ensure smooth layer-wise dataflow and efficient hardware mapping, we prioritize configurations where $m = k$, maintaining a unified activation format across all layers and avoiding costly inter-layer data reshaping, which will be discussed in detail in Section III-C.

B. Hardware View of AI Accelerator

In detail, Fig. 3 illustrates the hardware mapping of matrix multiplication in Gundam. Following the matrix decomposi-

tion strategy established earlier, the architecture is organized as a two-dimensional array of $n \times k$ MAC units. Each row of the array is connected to a dedicated activation buffer (A_Buffer#1 to A_Buffer#n), and each column is connected to a dedicated weight buffer (W_Buffer#1 to W_Buffer#k), resulting in a total of $(n + k)$ buffers. Gundam partitions large matrix multiplication tasks into a series of $n \times k$ and $k \times k$ submatrices, mapping each submatrix onto the PE array. Namely, we consistently set $m = k$ during decomposition to ensure that input and output data formats remain aligned across all layers. Thus, the size of each activation buffer is a multiple of $(n \times k)$, and the size of each weight buffer is a multiple of $(k \times k)$, supporting batch processing of multiple small submatrices. Besides, in ASICs, it may not be feasible to implement so many distributed buffers. Therefore, in practical designs, multiple buffers can be merged into a larger memory block to facilitate implementation.

During execution, the activations are read from activation buffers and distributed across the rows, while the weights are supplied from weight buffers to each column. Each MAC unit receives a corresponding activation and weight value per cycle, performing a MAC operation and generating an output that contributes to the final result matrix. This architecture allows for parallel computation of $n \times k$ outputs per cycle, achieving high throughput under appropriate resource allocation.

C. Resource Analysis for NN Deployment on Edge

Matrix partitioning is not only crucial for resource-constrained scenarios, but also helps maintain regularity and parallelism within practical hardware constraints. Algorithm 1 provides a systematic methodology for optimizing resource allocation in edge-AI matrix accelerators under tight hardware constraints. The procedure begins with the input matrix dimensions (N, M, K) and hardware parameters, including the available chip area (A_{chip}), area per PE (A_{PE}), buffer area per bit (A_{BUF}), data width (W_{data}), step size (G_{step}), and maximal PE array sizes ($PE_{\text{nmax}}, PE_{\text{kmax}}$). Besides, to achieve a more accurate estimation of area overhead, A_{BUF} includes the amortized area of the memory read/write control circuits. Similarly, A_{PE} accounts for the amortized area of the DMA and control logic within the accelerator. Notably, only 75% of the chip core area is used for resource analysis, with the remaining area reserved for place-and-route requirements and additional backend resources such as dummy cells and decap cells. Moreover, the PE array is typically limited to less than 50% of A_{chip}, reflecting the real-world case of resource distribution. Besides, the step size is chosen as a multiple of 8, consistent with practical hardware block granularity.

The algorithm enumerates all feasible (n, k) configurations (line#2) by stepping n and k from G_{step} up to their respective maxima. For each candidate, the activation and weight matrices are zero-padded so that M is a multiple of k, and all submatrix boundaries are aligned. This ensures that data size is the multiples of k (line#5, #6), facilitating smooth computation at block boundaries. Mathematically, this means each submatrix multiplication is $[n \times k] \cdot [k \times k]$, producing $n \times k$ outputs that directly feed into the next layer, eliminating the need for data reshaping and reducing hardware overhead as mentioned in Section III-A.

The next step (line#8) is to calculate the minimum buffer sizes required for activation functions (Buf_A^{min}) and weights (Buf_W^{min}). Here, we define that at least one complete $n \times k$ output block can be generated in local storage to avoid the need for buffer to store intermediate temporary data. If these minimum buffers cannot fit within the chip area limit, this configuration is not applicable.

Algorithm 1 Resource Analysis for Matrix Multiplication

Require: **Matrix Parameters:**
N, M, K: $\mathbf{A}_{N\times M} \cdot \mathbf{B}_{M\times K}$ {Matrix dimensions}
Hardware Parameters:
A_{chip}: Total chip core area budget $\times$ 0.75
A_{PE}: Area per PE (e.g., um^2/PE)
A_{BUF}: Area per memory buffer for activation/weight (e.g., um^2/bit)
W_{data}: Data bit width (e.g., 8-bit Integer)
G_{step}: Step size (e.g., 8 or 16)
$[PE_{\text{nmax}}, PE_{\text{kmax}}]$: Maximum PE array sizes along the n- and k-dimensions under the constraints of $0.5\times A_{\text{chip}}$, aligned to G_{step}
Ensure: Optimal configuration:
PE array structure (n,k), buffer allocation, and estimated throughput

1: Initialize $S_{\text{config}} = [\,]$
2: **Enumerate Feasible Configurations:**
3: **for** $n = G_{\text{step}}$ **to** PE_{nmax} **step** G_{step} **do**
4: **for** $k = G_{\text{step}}$ **to** PE_{kmax} **step** G_{step} **do**
5: $\mathbf{A}_{N'\times M'}$: Activation matrix after zero-padding, where $N' = n \times \lceil N/n \rceil$, $M' = k \times \lceil M/k \rceil$
6: $\mathbf{B}_{M'\times K'}$: Weight matrix after zero-padding, where $M' = k \times \lceil M/k \rceil$, $K' = k \times \lceil K/k \rceil$
7: $N_{\text{PE}} \leftarrow n \times k$ {Total MAC units}
8: **Calculate Minimum Storage Requirements:**
9: $Buf_A^{min} \leftarrow n \times M' \times W_{data}$ {Minimal activation buffer size}
10: $Buf_W^{min} \leftarrow k \times M' \times W_{data}$ {Minimal weight buffer size}
11: **if** $\left[(Buf_A^{min} + Buf_W^{min}) \times A_{\text{Buf}} + N_E \times A_{\text{PE}}\right] > A_{\text{chip}}$ **then**
12: **break** {No enough resource}
13: **end if**
14: **Search for Optimal Execution Time:**
15: $Buf_{total} \leftarrow (A_{\text{chip}} - N_{\text{PE}} \cdot A_{\text{PE}})/A_{\text{BUF}}$
16: Initialize $T_{best} = \infty$
17: Initialize $BW_{\text{mem}} = k * W_{data}(bits/cycle)$ {Bandwidth}
18: $T_{\text{comp}} \leftarrow N' \times M' \times K' \div N_{\text{PE}}$ {Computation time}
19: **for** $s = 1$ **to** $\lfloor Buf_{\text{total}}/Buf_A^{min} \rfloor$ **step** 1 **do**
20: $Buf_A^{current} \leftarrow s \times Buf_A^{min}$
21: $Buf_{remain} \leftarrow (Buf_{total} - Buf_A^{current})$
22: $Buf_W^{current} \leftarrow \lfloor Buf_{remain}/Buf_W^{min} \rfloor \cdot Buf_W^{min}$
23: **if** $Buf_W^{current} = 0$ **then**
24: **break** {No available buffer, stop allocation}
25: **end if**
26: $T_{\text{trans,act}} \leftarrow \frac{N'\times M'}{BW_{\text{mem}}} + \left\lceil \frac{K'/k}{Buf_W^{current}/Buf_W^{min}} \right\rceil \times \frac{N'/n}{s} \times \frac{Buf_W^{current}}{BW_{\text{mem}}}$ {Transfer time with activation reuse}
27: $T_{\text{trans,wt}} \leftarrow \frac{M'\times K'}{BW_{\text{mem}}} + \lceil \frac{K'/k}{Buf_W^{current}/Buf_W^{min}} \rceil \times \frac{N'/n}{s} \times \frac{Buf_A^{\text{current}}}{BW_{\text{mem}}}$ {Transfer time with weight reuse}
28: $T_{\text{trans,min}} \leftarrow \min(T_{\text{trans,act}}, T_{\text{trans,wt}})$ {Best strategy}
29: $T_{\text{total}} \leftarrow T_{\text{comp}} + T_{\text{trans,min}}$
30: **if** $T_{best} > T_{total}$ **then**
31: $T_{best} \leftarrow T_{\text{total}}$
32: **end if**
33: **end for**
34: **Append** $\{n, k, Buf_A^{current}, Buf_W^{current}, T_{best}\}$ to S_{config}
35: **end for**
36: **end for**
37: **Output:** S_{config} {Optimal configuration set}

The algorithm then searches all feasible splits of the remaining buffer between activations and weights (line#19). First, it computes the computation time (T_{comp}) based on total MAC units. Then, for each allocation, the data transfer time is evaluated for both activation-reuse (line#26) (where weights are more frequently updated) and weight-reuse (line#27) (where activations are more frequently updated) scenarios between external memory and internal memory. For each, the total transfer time is determined by the amount of data to be moved and the available bandwidth. The minimal transfer time across the two reuse strategies is selected, and combined with T_{comp} to yield the total expected execution time for that configuration.

Crucially, this evaluation enables the algorithm to select the reuse strategy that minimizes data movement and latency under resource constraints, which is particularly important for edge deployment. All valid configurations, including the PE array size (n, k), buffer allocation, and execution time, are collected in a candidate set S_{config} (line#34). The final output is the set of optimal resource allocations that best satisfy both hardware and workload requirements.

Algorithm 2 NN-Level Resource Scheduling Across Layers

Require: $\mathcal{L} = \{(N_i, M_i, K_i)\}_{i=1}^{L}$: Matrix shapes for all L layers in the model
Hardware parameters: Same as Algorithm 1
Ensure: Optimal shared configuration (n^*, k^*) across layers
Layer-wise buffer allocations, and total execution time

1: Initialize $S_{\text{model}} \leftarrow [\,]$ {Store all $S_{\text{config}}^{(i)}$}
2: **for** $i = 1$ to L **do**
3: Run **Algorithm 1** on (N_i, M_i, K_i)
4: Obtain $S_{\text{config}}^{(i)} \leftarrow \{(n, k, Buf_A, Buf_W, T)_j^{(i)}\}$
5: Append $S_{\text{config}}^{(i)}$ to S_{model}
6: **end for**
7: **Extract candidate shared** (n, k) **sets:**
8: $S_{\text{common}} \leftarrow \bigcap_{i=1}^{L} \left\{(n, k) \in S_{\text{config}}^{(i)}\right\}$
9: Initialize $T_{\text{model\_best}} \leftarrow \infty$
10: **for** each $(n, k) \in S_{\text{common}}$ **do**
11: $T_{\text{total}} \leftarrow 0$
12: **for** $i = 1$ to L **do**
13: Select config $(n, k, Buf_A, Buf_W, T) \in S_{\text{config}}^{(i)}$ with lowest T under fixed (n, k)
14: $T_{\text{total}} \leftarrow T_{\text{total}} + T$
15: **end for**
16: **if** $T_{\text{total}} < T_{\text{model\_best}}$ **then**
17: $T_{\text{model\_best}} \leftarrow T_{\text{total}}$
18: $(n^*, k^*) \leftarrow (n, k)$
19: Store best config for each layer: $S_i^* \leftarrow$ corresponding (Buf_A, Buf_W) for layer i
20: **end if**
21: **end for**
22: **Output:** (n^*, k^*), $\{S_i^*\}_{i=1}^{L}$, $T_{\text{model\_best}}$

Furthermore, Algorithm 2 extends the resource analysis model to accommodate entire NN models consisting of multiple distinct layers, each characterized by its own matrix dimensions. Since optimal hardware resource allocation for individual layers may differ significantly, Algorithm 2 performs a global analysis to harmonize configurations across all layers (line#2). It takes as input a set of matrix shapes (N_i, M_i, K_i) for each of the L layers in the model, and individually executes Algorithm 1 on each layer to produce respective candidate configuration sets ($S_{\text{config}}^{(i)}$).

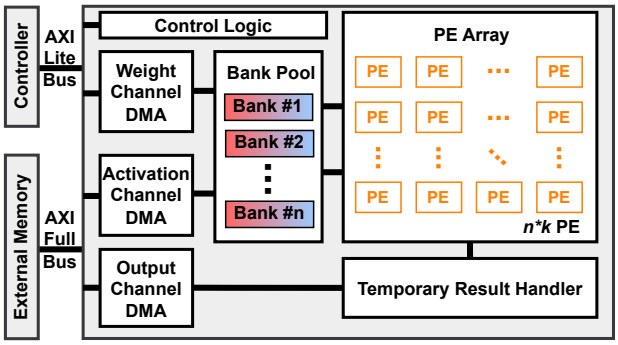

Fig. 4. AI Accelerator Architecture Based on NVDLA.

Subsequently, the algorithm identifies common configurations (n, k) shared across all layers, ensuring consistent hardware dimensions for efficient resource reuse (line#10). For each candidate common configuration, the algorithm selects the best-performing memory allocation from each layer-specific candidate set to achieve minimum total NN execution time. By summing these layer-wise optimal execution times (line#13, #14), it estimates the total execution latency for the entire model. After evaluating all common configurations, the algorithm selects the globally optimal configuration that minimizes the total model execution time (line#19). The final output includes the selected optimal PE array sizes (n^*, k^*), associated optimal buffer allocations for each layer, and the expected overall execution latency, thus enabling efficient hardware deployment for edge-AI applications.

IV. Gundam-guided AI Accelerator Realization

A. Hardware Characterization and Baseline Setup

To evaluate the effectiveness of Gundam, we use the open-source NVDLA as a baseline architecture. As shown in Fig. 4, the PE array size and organization are determined by the resource allocation results from Gundam. The bank pool provides shared on-chip memory, where each bank can be dynamically assigned to either activation or weight storage on a per-layer basis, according to the optimal partitioning determined by configurations generated from the aforementioned algorithms in Gundam. Moreover, dedicated DMA engines manage data movement between external memory and the bank pool for weights, activations, and outputs, while control logic oversees all internal operations. Temporary results are handled by a dedicated temporary result handler, ensuring smooth accumulation and data flow.

On the other hand, we conduct detailed hardware characterization to support accurate resource modeling and design-space exploration. Key parameters, such as area per PE (A_{PE}), and buffer area per bit (A_{BUF}), are extracted based on logic synthesis with standard-cell libraries and memory compilers. To validate the scalability of Gundam, these parameters are profiled under three CMOS technology nodes (16/22/28nm).

In addition, a set of representative transformer-based models (i.e., ViT-Tiny/Small/Base) is selected as benchmark workloads. These models cover a wide spectrum of matrix multiplication shapes and memory access patterns, making them

TABLE I
DESIGN INPUTS AND CONSTRAINTS USED IN GUNDAM EVALUATION

Metric	16nm	22nm	28nm
Chip core area budget (mm^2)	1.4	1.8	0.8
Power Supply (V)	0.8	0.8	0.8
Frequency (MHz)	700	600	500
Target model	ViT-Tiny/Small/Base		

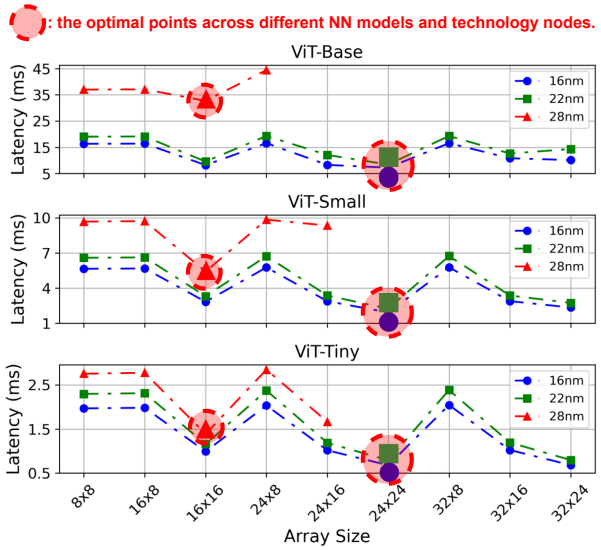

Fig. 5. Design Space Exploration of AI Accelerator Configurations.

suitable for evaluating both the configurability and analytical accuracy of the proposed design and analysis model.

B. Hardware Implementation Guided by Gundam

Based on the NVDLA architecture, the target models and the chip area budgets listed in Table I for each technology node, we design AI accelerators in 16nm, 22nm and 28nm CMOS processes. For each case, we perform an exhaustive search for the optimal execution time by incrementing the design parameters in steps of $G_{\text{step}} = 8$. As illustrated in Fig. 5, the search space covers a range of PE array sizes from 8×8 up to 32×24, ensuring comprehensive evaluation of all feasible configurations for ViT-Tiny, ViT-Small, and ViT-Base models. Through the Gundam design space exploration, we identify that the optimal PE array size for both 16nm and 22nm nodes is 24×24, while for the 28nm node, the best configuration is 16×16, each accompanied by their respective memory parameters. These optimal configurations are selected based on minimum inference time across the full search range. Besides, thanks to the lightweight and analytical nature of Gundam, the entire search for optimal configurations on a standard GPU platform completes within just a few seconds, enabling rapid iteration and early-stage design exploration.

Then, we construct layouts for each technology node using the selected array sizes and buffer specifications from Gundam. The layouts of these designs are shown in Fig. 6. The layouts of these designs are implemented via a standard digital IC flow, with logic synthesis performed by Synopsys Design Compiler and physical design completed in Cadence Innovus.

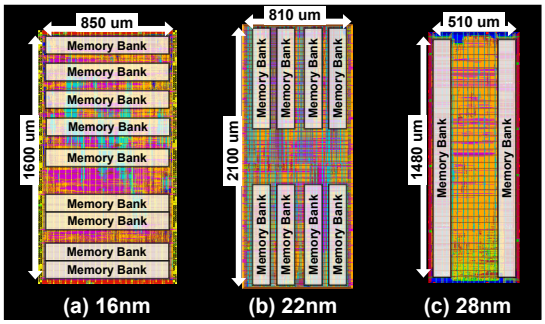

Fig. 6. Layout of Designs Across Different Technology Nodes.

TABLE II
VALIDATION OF GUNDAM ESTIMATES ACROSS TECHNOLOGY NODES

Metric	16nm	22nm	28nm
Optimal (n, k)	(24, 24)	(24, 24)	(16, 16)
Total Buffer Size (KB)	540	456	180
Total Core Area (mm^2)	1.360	1.701	0.755
Estimation Error (Area, %)	2.94	5.82	5.96

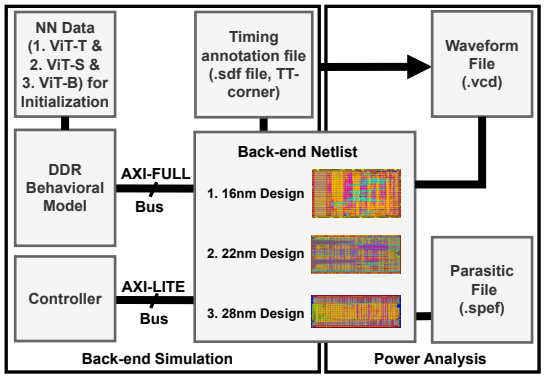

Fig. 7. Benchmark Setup for Performance Analysis.

Table II summarizes the estimation results of Gundam model across three technology nodes. To validate its accuracy, we implement the corresponding configurations using RTL synthesis and back-end place-and-route, extracting post-layout information. The Gundam-estimated values closely align with actual post-layout results, with area size estimation errors below 6%. This demonstrates that Gundam achieves high-fidelity hardware-level estimation, making it well suited for early-stage accelerator design across different process nodes.

V. PERFORMANCE ANALYSIS

A. Benchmark Setup

The benchmark setup is shown in Fig. 7. Post-layout simulation is performed in Synopsys VCS using extracted delay information (.sdf file, TT-corner) from the physical layout. Three representative models, ViT-Tiny/Small/Base, are used for evaluation. During initialization, the testbench loads model data into a DDR behavioral model, and the controller configures on-chip registers via the AXI-Lite bus to activate the accelerator. Data exchange with DDR memory occurs over the AXI-Full bus during execution. After simulation, a waveform file (.vcd) is generated by VCS. The parasitic file (.spef) extracted from layout and the waveform file are used

TABLE III
PERFORMANCE EVALUATION AMONG DIFFERENT NNS

Metric		16nm	22nm	28nm
ViT-T[1]	Inference Time (μs)	677.90	790.88	1387.13
	Peak Perf. (TOPS/W)	11.86	9.13	7.34
	Average Perf. (TOPS/W)	9.31	7.15	5.63
ViT-S[2]	Inference Time (μs)	1921.66	2241.93	5409.65
	Peak Perf. (TOPS/W)	11.78	9.22	7.21
	Average Perf. (TOPS/W)	9.18	7.22	5.70
ViT-B[3]	Inference Time (μs)	7200.30	8400.35	32697.70
	Peak Perf. (TOPS/W)	11.90	9.08	7.18
	Average Perf. (TOPS/W)	9.28	7.20	5.66
Peak Energy Efficiency[4] (TOPS/W)		16.58	12.81	10.08

[1]15.3%/57.4% sparsity for weight/activation, accuracy: 76.82%.
[2]19.6%/64.9% sparsity for weight/activation, accuracy: 81.79%.
[3]22.4%/79.2% sparsity for weight/activation, accuracy: 82.25%.
[4]50% sparsity, 100% PE utilization.

TABLE IV
COMPARISON WITH EXISTING ANALYTICAL COST MODEL

Statistics	MAESTRO [12]	Timeloop [13]	CoSA [15]	This work
Description	Data-reuse-based Model	Throughput-based Model	Optimization-based Model	Resource-Latency-based Model
Data Reuse	Yes	Yes	Yes	**Yes**
Latency	Yes	No[1]	Yes	**Yes**
Throughput	Yes	Yes	Yes	**Yes**
NN Mapping	Yes	Yes	Yes	**Yes**
PE Size Search	No	No	No	**Yes**[2]
Memory Size Search	No	No	No	**Yes**[2]

[1]Used exclusively for performance evaluation; not in analytical modeling.
[2]Automatically searches for the optimal AI accelerator configurations.

as input to Synopsys PrimeTime PX for accurate post-layout power analysis. This flow ensures realistic evaluation of both performance and power characteristics for the designs.

B. Performance Evaluation

Table III presents the performance evaluation results of the proposed accelerator under three technology nodes (16/22/28nm) and three representative transformer models: ViT-Tiny/Small/Base. Across all process nodes, inference time increases with model size and older technology. For example, ViT-Tiny achieves an inference time of 678 μs at 16nm, while ViT-Base reaches 32697 μs at 28nm. Peak and average performance metrics are highest at 16nm and decrease at larger nodes, reflecting expected trade-offs between energy efficiency and fabrication technology. Peak performance reaches up to 11.86 TOPS/W for ViT-Tiny at 16nm, with corresponding average performance of 9.31 TOPS/W. Even under larger models and older nodes, the design sustains competitive efficiency (e.g., 5.66 TOPS/W average for ViT-Base at 28nm).

Compared with SOTA accelerator designs in [17], which achieve over 50 TOPS/W by employing aggressive quantization (e.g., binary precision) and compute-in-memory techniques, our Gundam-guided designs adopt a more balanced and versatile approach. While those ultra-efficient designs excel in peak metrics, they typically require highly customized hardware and offer limited flexibility for diverse AI workloads. In contrast, Gundam focuses on model-aware resource allocation and structured matrix decomposition, enabling robust and predictable performance across full NN execution rather than isolated layers. Our designs, implemented in 16/22/28nm processes, consistently achieve 10–20 TOPS/W energy efficiency under standard operating conditions (0.8V) without voltage or frequency scaling. Notably, further scaling to lower voltage and frequency can increase energy efficiency by several times. This demonstrates Gundam's capability to bridge theoretical optimization and practical deployment, providing a scalable solution with strong performance-efficiency trade-offs on edge.

C. Comparison with Existing Models

Table IV compares representative accelerator modeling and scheduling models. MAESTRO adopts a data-reuse-based analytical model, providing fast estimation of latency and energy for a given mapping, but lacks automated architecture search capabilities. Timeloop uses a throughput-based model with an iterative mapper, enabling flexible exploration of mappings under fixed hardware, but its random or brute-force search is often time-consuming and may miss global optima in large design spaces. CoSA formulates DNN scheduling as a mixed-integer programming problem, supporting one-shot operator-level schedule optimization, yet its search space is limited by the accuracy and completeness of constraint modeling. In contrast, Gundam employs a resource-latency-based model and supports unified, multi-granularity search across PE size, memory size, and architectural parameters under resource constraints. This enables Gundam to efficiently and deterministically find optimal solutions for AI acclerator configurations, covering a much wider design space.

VI. CONCLUSION

In this paper, we propose Gundam, a lightweight and unified analytical model designed for agile and accurate resource estimation and allocation in edge-AI matrix accelerators. Gundam models matrix operations to enable structured estimation of PE array dimensions, memory size, buffer partitioning, and data reuse strategies. It supports rapid design space exploration across various matrix shapes and technology nodes, achieving less than 6% deviation between analytical estimates and post-layout hardware results. Unlike existing models that assume fixed architectures or focus only on peak performance, Gundam can automatically search for optimal accelerator configurations under chip area and bandwidth constraints. This offers a practical and scalable solution for early-stage design planning and efficient deployment of AI accelerators on resource-limited edge platforms.

ACKNOWLEDGMENT

This work is partially supported by the Grant-in-Aid for Scientific Research (S) from Japan Society for the Promotion of Science (JSPS) under Grant 24H00073, National Natural Science Foundation of China under Grant No. 62274081, and Grant no. 2023QN10X177.

REFERENCES

[1] T. Mohaidat and K. Khalil, "A Survey on Neural Network Hardware Accelerators," in IEEE Transactions on Artificial Intelligence, vol. 5, no. 8, pp. 3801-3822, Aug. 2024, doi: 10.1109/TAI.2024.3377147.

[2] Fawzi, Alhussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning," in Nature, doi: 10.1038/s41586-022-05172-4.

[3] N. Zhang, S. Ni, L. Chen, T. Wang and H. Chen, "High-Throughput and Energy-Efficient FPGA-Based Accelerator for All Adder Neural Networks," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2025.3543213.

[4] M. M. H. Shuvo, S. K. Islam, J. Cheng and B. I. Morshed, "Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review," in Proceedings of the IEEE, vol. 111, no. 1, pp. 42-91, Jan. 2023, doi: 10.1109/JPROC.2022.3226481.

[5] D. Kudithipudi et al., "Design principles for lifelong learning AI accelerators." Nature Electronics 6.11 (2023): 807-822.

[6] A. Feldmann, C. Golden, Y. Yang, J. S. Emer and D. Sanchez, "Azul: An Accelerator for Sparse Iterative Solvers Leveraging Distributed On-Chip Memory," 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), Austin, TX, USA, 2024, pp. 643-656, doi: 10.1109/MICRO61859.2024.00054.

[7] C. Yi, S. Jian, Y. Tan and Y. Zhang, "HMO: Host Memory Optimization for Model Inference Acceleration on Edge Devices," 2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Kuching, Malaysia, 2024, pp. 2813-2819, doi: 10.1109/SMC54092.2024.10831215.

[8] A. Symons, L. Mei and M. Verhelst, "LOMA: Fast Auto-Scheduling on DNN Accelerators through Loop-Order-based Memory Allocation," 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), Washington DC, DC, USA, 2021, pp. 1-4, doi: 10.1109/AICAS51828.2021.9458493.

[9] Q. Cheng et al., "A 13-34 TOPS/W Edge-AI Processor Featuring Booth-Value-Confined Accelerator, Near-Memory Computing, and Contiguity-Aware Mapping," 2024 IEEE Asian Solid-State Circuits Conference (A-SSCC), Hiroshima, Japan, 2024, pp. 1-3, doi: 10.1109/A-SSCC60305.2024.10849341.

[10] S. Nag, G. Datta, S. Kundu, N. Chandrachoodan and P. A. Beerel, "ViTA: A Vision Transformer Inference Accelerator for Edge Applications," 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 2023, pp. 1-5, doi: 10.1109/ISCAS46773.2023.10181988.

[11] M. Huang et al., "EdgeLLM: A Highly Efficient CPU-FPGA Heterogeneous Edge Accelerator for Large Language Models," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 72, no. 7, pp. 3352-3365, July 2025, doi: 10.1109/TCSI.2025.3546256.

[12] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer and A. Parashar, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," in IEEE Micro, vol. 40, no. 3, pp. 20-29, 1 May-June 2020, doi: 10.1109/MM.2020.2985963.

[13] A. Parashar et al., "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 2019, pp. 304-315, doi: 10.1109/ISPASS.2019.00042.

[14] Suhail Basalama, and Jason Cong, "Stream-HLS: Towards Automatic Dataflow Acceleration," In Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25), New York, NY, USA, 103–114. https://doi.org/10.1145/3706628.3708878

[15] Q. Huang et al., "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2021, pp. 554-566, doi: 10.1109/ISCA52012.2021.00050.

[16] X. Wang et al., "Optimizing Edge AI: A Comprehensive Survey on Data, Model, and System Strategies," arXiv:2501.03265, Jan. 2025.

[17] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, T. Xie, S. Han, Y. Xie, P. Debacker, M. Verhelst, Y. Wang. "Neural Network Accelerator Comparison" [Online]. Available: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/