FADiff: Fusion-Aware Differentiable Optimization for DNN Scheduling on Tensor Accelerators

Shuao Jia[1], Zichao Ling[1], Chen Bai[2], Kang Zhao[1], Jianwang Zhai[1]

[1]Beijing University of Posts and Telecommunications [2]The Hong Kong University of Science and Technology

Corresponding author: zhaijw@bupt.edu.cn

Abstract

Efficient deployment of Deep Neural Networks (DNNs) —such as Large Language Models (LLMs)—on tensor accelerators is essential for maximizing computational efficiency in modern AI systems. However, achieving this is challenging due to the enormous and complex design space created by the interaction of intra-layer mapping and inter-layer fusion. In this work, we present FADiff, a gradient-based optimization framework capable of automatically identifying high-quality intra-layer mapping and inter-layer fusion strategies to accelerate inference for DNN workloads. We first construct a unified and differentiable analytical cost model, which accurately predicts the energy and latency of both single-layer mappings and various layer fusion strategies. Then, by encoding discrete constraints into the loss function, we employ a gradient-based approach to efficiently explore the vast design space, determining the optimal joint strategy for mapping and fusion. Experimental results demonstrate the superiority of FADiff, achieving better optimization in terms of energy and latency compared to existing methods.

1 Introduction

Efficient deployment of DNNs on resource-constrained hardware, such as edge tensor accelerators, is a critical step for enabling high-performance on-device intelligence. A central challenge in this process lies in bridging the gap between the high-level model and the low-level hardware execution. This requires exploring a vast combinatorial design space of deployment configurations, where intra-layer mapping (i.e., how computations and data within a single layer are tiled and mapped onto hardware resources) and inter-layer fusion (i.e., how multiple adjacent layers are grouped and executed jointly) interact in non-trivial ways. These decisions jointly determine data reuse, memory traffic, and ultimately overall energy and latency efficiency. As models grow deeper and accelerators become more complex, this coupled mapping-fusion design space becomes increasingly difficult to reason about and optimize.

Existing studies on DNN deployment design space exploration can be broadly categorized into heuristic and learning-based approaches. Heuristic frameworks such as TVM [1], DeFiNES [2], SoMa [3], and Klotski [4, 5] rely on combinatorial or search-based optimization to traverse the vast design space. Learning-based efforts, including Active Learning-guided optimization [6] and Deep Gaussian Transfer Learning [7], aim to improve efficiency through surrogate modeling and experience transfer. Despite their differences, both paradigms fundamentally depend on iterative surrogate evaluation and empirical search, resulting in slow convergence, limited interpretability, and a lack of theoretical grounding.

More recently, gradient-based optimization has emerged as an effective paradigm for accelerating DNN deployment design space exploration. The key insight, pioneered by DOSA [8] and Felix [9], lies in formulating differentiable energy and latency models. This

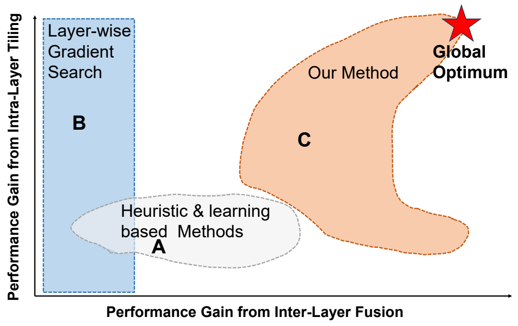
Layer-wise Gradient Search
B
Our Method
Global Optimum
C
Heuristic & learning based Methods
A
Performance Gain from Intra-Layer Tiling
Performance Gain from Inter-Layer Fusion

Figure 1: The joint design space of inter-layer Fusion and intra-layer Mapping. While traditional heuristic and learning-based methods (A) explore a limited sub-region and layer-wise gradient approaches (B) are confined to a single axis, our work (C) performs a comprehensive, differentiable search across the entire space to approach the global optimum (red ⋆).

transformation converts the discrete, combinatorial design space into a continuous domain, thereby enabling efficient gradient-based optimization. Rather than relying on heuristic or learning-based algorithms, this paradigm leverages analytical gradients in a theoretically grounded manner, resulting in improved optimization efficiency and interpretability by explicitly modeling parameter sensitivities.

However, these methods remain confined to layer-wise optimization. They do not address the fundamental and tightly coupled problem of inter-layer fusion within the same differentiable framework. Consequently, layer fusion, which determines how operations are merged together to reduce expensive off-chip memory traffic, is still handled separately by heuristic or learning-based methods. This prevents a truly global optimization of both the inter-layer fusion strategy and the intra-layer mapping. As illustrated in Figure 1, this reveals a fundamental gap in the existing state-of-the-art approaches:

(1) While general in principle, heuristic and learning-based methods face severe scalability barriers when addressing the combinatorially large, high-dimensional design space of joint mapping and fusion. For instance, Gaussian-process–based Bayesian Optimization (BO) requires inverting an $N \times N$ covariance matrix, incurring $O(N^3)$ cost in the number of evaluated samples N; in high-dimensional design spaces, N must grow rapidly to explore the space, making BO quickly intractable.
(2) Existing gradient-based approaches, though computationally efficient, are inherently limited to intra-layer optimization. Their fundamental constraint stems from the non-differentiable nature of inter-layer fusion decisions. Fusion involves discrete, structural choices—such as determining which layers to combine—that disrupt the continuous optimization space required for standard backpropagation. As a result, these

methods typically assume layer independence, optimizing each layer in isolation under a simplified assumption.

To bridge this gap, we propose FADiff, a gradient-based optimization framework that, to our knowledge, is the first to unify intra-layer mapping and inter-layer fusion into a single, jointly differentiable optimization problem, as illustrated in Figure 2. At the heart of FADiff lies a differentiable analytical cost model that captures the complex interaction between mapping and fusion strategies in a unified form. This unified formulation enables efficient gradient-based co-optimization of both problems, yielding high-performance and hardware-valid DNN deployment solutions. The main contributions are as follows:

- We develop an analytical cost model that jointly captures intra-layer mapping and inter-layer fusion behaviors, enabling a unified differentiable formulation for energy and latency trade-offs on tensor accelerators.
- We encode discrete constraints into the loss function, allowing gradient-based optimization that converges quickly and delivers both interpretable and valid deployment solutions.
- We demonstrate that FADiff achieves an average of 15% reduction in Energy-Delay Product (EDP) compared with state-of-the-art layer-wise gradient-based approaches.

The rest is organized as follows. Section 2 gives the preliminaries of this work. Section 3 then details the proposed fusion-aware differentiable optimization framework, followed by an experimental evaluation in Section 4. Finally, Section 5 concludes this paper.

2 Preliminaries

This section introduces the target hardware, the layer mapping and fusion strategies, and the formal problem formulation.

2.1 Target Hardware Architecture

Our work targets Gemmini [10], an open-source tensor accelerator for DNN inference (Figure 2(a)). It uses a 2D systolic array to accelerate matrix multiplication (GEMM) and convolution (CONV), with inputs either broadcast or streamed systolically across the array. An on-chip scratchpad and accumulator, both software-managed, provide flexible control over data movement.

During execution, operands are fetched from DRAM to the scratchpad, processed by the systolic array, and accumulated before being written back. A weight-stationary (WS) dataflow is adopted to maximize weight reuse and reduce memory traffic.

Figure 2(a) annotates the bandwidth and the energy-per-access (EPA) of each memory level; for on-chip buffers, we model EPA using a small multi-layer perceptron (MLP) as a function of buffer capacity.

2.2 Layer Mapping and Layer Fusion

Deploying a DNN on a tensor accelerator requires exploring a vast design space. This work focuses on efficiently optimizing two tightly coupled design strategies: Layer Mapping and Layer Fusion.

Layer Mapping defines how computations within each layer are scheduled and executed on the tensor accelerator, including both spatial and temporal mapping, as depicted in Figure 2(c). Spatial mapping assigns computations to the Processing Elements (PEs) of the systolic array, determining hardware utilization and spatial parallelism. Temporal mapping governs data tiling and movement across memory levels, which directly impacts data reuse and bandwidth efficiency.

Layer Fusion combines multiple dependent layers into a single execution pipeline, forming a composite computational kernel that minimizes costly off-chip DRAM accesses. By keeping intermediate tensors on-chip, it effectively reduces both energy consumption and latency. Fusion is feasible only when layers have (1) direct producer-consumer dependencies, (2) compatible tensor shapes, and (3) fit within the available on-chip buffer capacity.

However, jointly optimizing mapping and fusion remains challenging because of their tight coupling and the combinatorial explosion of design spaces. Heuristic and learning-based frameworks [2–7] explore the design space through search-based or surrogate modeling methods, but both depend heavily on empirical evaluations and extensive search. Differentiable frameworks such as DOSA [8] are confined to layer-wise mapping and do not integrate fusion into the differentiable formulation.

2.3 Problem Formulation

The joint optimization of layer mapping and fusion is formulated. We model the DNN as a Directed Acyclic Graph (DAG), $G = (V, E)$. In this representation, each vertex $v \in V$ corresponds to a computational layer, and each directed edge $e \in E$ signifies the tensor dependency between these layers. This graph-based model explicitly captures the producer-consumer relationships that are essential for optimizing layer mapping and fusion.

Its key optimization constraints are summarized below, as they fundamentally shape the optimization space. These parameters include: (1) the number of PEs, which determines the available computational parallelism; (2) the limited capacity of on-chip memory buffers, which governs data residency and reuse; and (3) the strict compatibility requirements on tile shapes across adjacent layers.

A deployment strategy specifies the spatio-temporal execution of the DNN on hardware through intra-layer mapping and inter-layer fusion. Our goal is to choose the strategy that minimizes EDP subject to the above constraints.

3 Methodology

Existing exploration for DNN deployment on tensor accelerators is hampered by a dichotomy: computationally expensive, non-differentiable combinatorial searches on one hand, and differentiable methods limited to layer-wise optimization that ignore fusion opportunities on the other. As shown in Figure 2, our work bridges this gap by introducing a unified and differentiable framework for co-optimizing layer mapping and fusion.

3.1 Continuous Representation of Strategies

A core challenge in gradient-based exploration is that both mapping and fusion decisions are inherently *discrete*, preventing direct optimization through gradient descent. To make the design space continuous and differentiable, we introduce unified parameterizations for mapping and fusion strategies.

3.1.1 Mapping Strategy. Each layer's mapping is characterized by its tiling factors across all problem dimensions and memory levels. We adopt a unified 7-dimensional problem space, N, K, C, P, Q, R, S, where N represents the batch size, K and C denote the output and input channel depths, P and Q are the output feature map spatial dimensions, and R and S correspond to the kernel filter height and width, respectively. This formulation generalizes both CONV and GEMM operations (with $P, Q, R, S = 1$ for GEMM). The mapping targets a 4-level memory hierarchy consisting of $L3$ (DRAM), $L2$

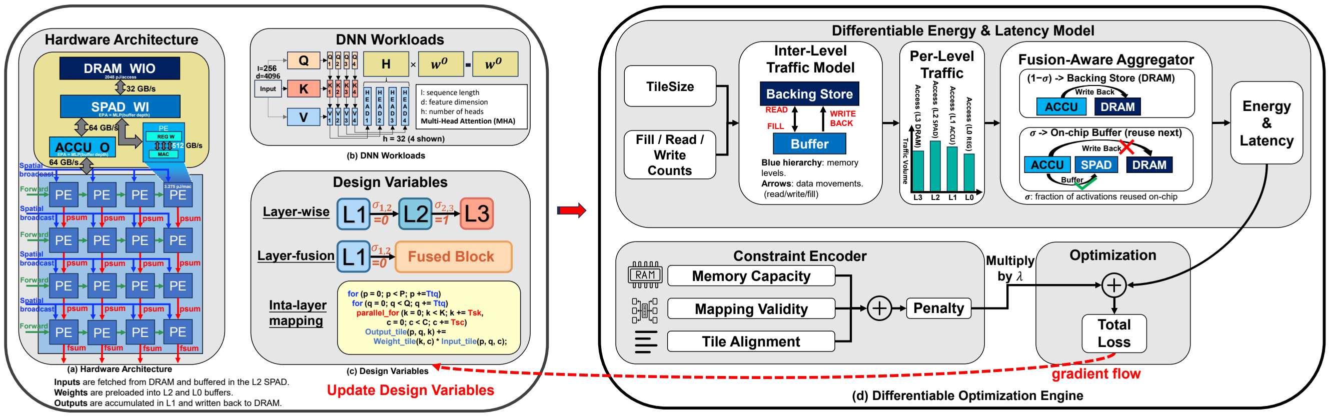

Figure 2: Overview of FADiff. Left (regions a–c): hardware architecture, DNN workloads, and design variables. Right (region d): differentiable optimization engine.

(Scratchpad), $L1$ (Accumulator), and $L0$ (PE Registers). The complete set of temporal tiling factors is denoted by $T_{t,d,m}$, where $T_{t,d,m}$ represents the temporal partition size of dimension d at memory level m. Similarly, the spatial tiling factors are denoted by $T_{s,d,m}$, specifying how each dimension is distributed spatially across the PE array. For notational simplicity, we use $T_{d,m}$ to denote a generic tiling factor (either temporal or spatial).

While gradient-based optimization requires the mapping factors $T_{t,d,m}$ and $T_{s,d,m}$ to be continuous, the underlying design space is inherently discrete and integer-valued. To enable differentiable optimization over such inherently discrete choices, we introduce a continuous relaxation based on the Gumbel-Softmax reparameterization trick [11].

For a given problem dimension of size n, we must select an integer tiling factor. The set of valid tiling candidates, denoted as $\{d_j\}_{j=1}^{K}$, consists of the integer divisors of n, where K is their total count. To map from the continuous representation $T_{d,m}$ to a differentiable selection over these discrete values, each divisor d_j is assigned a logit l_j reflecting its proximity:

$$l_j = -\alpha(T_{d,m} - d_j)^2, \qquad \text{where } \alpha > 0. \tag{1}$$

A differentiable categorical distribution over d_j is then obtained via the Gumbel–Softmax transformation:

$$p_j = \frac{\exp\left((l_j + g_j)/\tau\right)}{\sum_{k=1}^{K} \exp\left((l_k + g_k)/\tau\right)}, \qquad g_j \sim \text{Gumbel}(0, 1), \tag{2}$$

with temperature τ gradually annealed from τ_0 to $\tau_{\min}$ during optimization. The expected discrete selection is expressed as:

$$\hat{d} = \sum_{j=1}^{K} p_j\, d_j. \tag{3}$$

As $\tau \longrightarrow 0$, the probability distribution collapses to a one-hot vector, and $\hat{d}$ converges to $d^* = \arg\max_j l_j$. A straight-through gradient estimator [12, 13] is employed so that the forward pass yields discrete selections while the backward pass preserves differentiability, thereby enabling end-to-end optimization of integer tiling factors within the continuous search space.

3.1.2 *Fusion Strategy*

The decision to fuse two adjacent nodes v_i and v_{i+1} in the computation graph is modeled by a continuous, learnable variable $\sigma_i \in [0, 1]$. Values of σ_i close to 1 indicate fusion (on-chip reuse), while σ_i near 0 correspond to separation (DRAM write-back). This continuous relaxation bridges discrete fusion boundaries and allows fusion decisions to be optimized jointly with mapping.

Unlike mapping, fusion is handled as a post-optimization decision: the continuous variable σ_i is directly optimized during the search, and its discrete state is determined after optimization. Thus, no discrete decision mechanism such as projection or sampling is required during optimization.

3.1.3 *Unified Representation*

The overall set of trainable parameters for the DNN thus consists of the mapping factors $\{T_{t,d,m}\}$ and $\{T_{s,d,m}\}$, and the fusion variables $\{\sigma_i\}$, forming a differentiable representation that unifies mapping and fusion in a single search space.

3.2 Differentiable Energy and Latency Model

To enable our gradient-based exploration, we developed a unified and differentiable cost model. This model estimates the end-to-end energy and latency for a complete deployment strategy.

3.2.1 *Data Traffic*

Data movement is a principal contributor to the overall latency and energy consumption in tensor accelerators, with its energy cost often rivaling or even exceeding that of computation [14]. We formulate the total inter-level data movement as a differentiable function of the continuous mapping parameters. Critically, we also model the impact of layer fusion, where a continuous variable σ_i governs on-chip data reuse and its associated traffic. This ensures the entire data-driven cost is differentiable, enabling gradient propagation through the memory hierarchy.

(1) Fill Traffic. For a tensor $T \in \{I, W\}$ (I is input tensor, W is weight tensor), the fill traffic to L_i is defined as:

$$\text{Fill}(L_i, T) = \text{TileSize}(i, T) \cdot \text{FetchCount}(i, T), \tag{4}$$

where the tile size (at L_i) and its fetch count are given by:

$$\text{TileSize}(i, T) = \prod_{d} \prod_{k=0}^{i} T_{t,d,k} \cdot T_{s,d,k}, \tag{5}$$

$$\text{FetchCount}(i, T) = \prod_{d} \prod_{k=i+1}^{M} T_{t,d,k}. \tag{6}$$

The outer product $\prod_d$ ranges over all problem dimensions d associated with tensor T (e.g., *RSCK* for weights), and L_M denotes the highest memory level in the hierarchy (e.g., DRAM).

(2) Read Traffic. Read traffic quantifies the data transferred from a lower-level (slower) memory to a higher-level (faster) one, initiated by a request from the higher level. We distinguish two fundamental types of read operations based on the request's origin and the transfer characteristics: inter-memory reads, which involve transferring entire data tiles between memory levels, and PE-supplying reads, which provide data for direct consumption by the PE array. Tensor T is in $\{I, W\}$, where I is the input tensor and W is the weight tensor.
(a) Inter-Memory Reads. Transfers between memory levels (e.g., from DRAM L_3 to on-chip scratchpad L_2) are modeled as Inter-Memory Reads. The total read traffic for a tensor T at level L_{i+1} is defined as:

$$\mathrm{Read}(L_{i+1}, T) = \mathrm{TileSize}(i, T) \cdot \mathrm{FetchCount}(i, T). \tag{7}$$

Broadcast reuse does not apply to these transfers, as there is no broadcast network between these memory levels.
(b) PE-Supplying Reads. Reads from the innermost buffer L_i (i.e., the buffer level directly supplying the PEs) feed the PE array. The total traffic is proportional to the number of MAC operations (Ops) and inversely proportional to the spatial broadcast factor Bcast_T:

$$\mathrm{Read}(L_\mathrm{i}, T) = \frac{\mathrm{Ops}}{\mathrm{Bcast}_T}. \tag{8}$$

Here, Ops is the total number of multiply-accumulate operations for the layer, and Bcast_T is the spatial broadcast factor for tensor T, defined as:

$$\mathrm{Bcast}_T = \prod_{d \notin \mathrm{dims}(T)} T_{s,d,m}. \tag{9}$$

(3) Write-Back Traffic. Write-back traffic models the movement of output data (tensors $T \in \{O\}$) originating from the PE array. This data flow is constrained: output tensors and their partial sums are only stored in the L1 Accumulator and L3 DRAM, bypassing the L2 Scratchpad and L0 Registers. This architecture leads to two distinct kinds of write-back operations. The first, inter-memory write-back, is the transfer of completed output tiles from a higher-level buffer (i.e., L1) to a lower-level memory (i.e., L3 DRAM). The second, accumulation write-back, is the initial transfer of computed results from the PE array into the L1 Accumulator.
(a) Inter-Memory Write-back. This models the write-back traffic from a higher-level memory to a lower-level one (e.g., from the L1 accumulator to the L3 DRAM).

$$\mathrm{WriteBack}(L_i, T) = \mathrm{TileSize}(i, T) \cdot \mathrm{WriteCount}(i, T). \tag{10}$$

Here, $\mathrm{WriteCount}(i, T)$ represents the number of times the tile at L_i is written back. Similar to the fill traffic model as defined earlier in Equation (6), this count is determined by the iterations of the outer temporal loops.
(b) Accumulation Write-back. The write traffic to the L1 accumulator for an output tensor T is given by:

$$\mathrm{WriteBack}(L_1, T) = \frac{\mathrm{Ops}}{\mathrm{Reduce}_T}. \tag{11}$$

Here, Ops is the total MAC operations. Reduce_T is the spatial reduction factor, reflecting how partial sums are accumulated directly within the PE array (e.g., via a spatial reduction network or forward-and-reduce) before being written to L_1. The spatial reduction factor Reduce_T is defined as:

$$\mathrm{Reduce}_T = \prod_{d \notin \mathrm{dims}(T)} T_{s,d,m}. \tag{12}$$

(4) Fusion-Aware Boundary. The boundary that governs data movement between on-chip and off-chip memory is continuously modulated by the fusion variable, allowing a smooth, differentiable shift between writing results back to DRAM and directly reusing them on-chip. Rather than sending all intermediate results back to DRAM, part of the output produced by layer or node v_i can stay on-chip, ready for immediate reuse by the next layer or node v_{i+1}.

This mechanism is driven by a continuous fusion variable $\sigma_i \in [0, 1]$, which smoothly spans the spectrum between two limiting cases: a non-fusion regime ($\sigma_i \approx 0$), where every piece of data is written back to DRAM, and a full-fusion regime ($\sigma_i \approx 1$), where intermediate outputs stay entirely in on-chip buffers. As σ_i increases, the amount of data written back off-chip from $L_1 \rightarrow L_3$ decreases proportionally, while additional on-chip data copy from $L_1 \rightarrow L_2$ are introduced to supply the next layer. At the same time, the fill traffic from DRAM $L_3 \rightarrow L_2$ for v_{i+1} is correspondingly decreased, reflecting the improved data locality enabled by fusion.

Formally, these relationships are expressed as: For layer or node v_i, let O denote the output tensor and $\mathrm{WriteBack}_0(L_3, O)$ and $\mathrm{Fill}_0(L_2, I)$ denote the baseline (non-fused) traffic without fusion:

$$\mathrm{WriteBack}(L_3, O) = (1 - \sigma_i)\, \mathrm{WriteBack}_0(L_3, O), \tag{13}$$

$$\mathrm{Copy}(L_1 \rightarrow L_2) = \sigma_i\, \mathrm{WriteBack}_0(L_3, O). \tag{14}$$

For next layer or node v_{i+1}, let I denote its input tensor:

$$\mathrm{Fill}(L_2, I) = (1 - \sigma_i)\, \mathrm{Fill}_0(L_2, I). \tag{15}$$

By modeling fusion with this continuous control variable, the model quantifies how inter-layer data reuse dynamically alters both off-chip and on-chip traffic, while preserving full differentiability for gradient-based optimization.

3.2.2 *Latency Model*

We adopt the roofline-style model that assumes computation and memory access can fully overlap when estimating latency. Under this assumption, the execution time of each layer is determined by whichever dominates—compute throughput or memory bandwidth. The overall latency is then obtained as the sum of all per-layer latencies.

The latency of layer or node $v_{i'}$ is modeled as:

$$\mathrm{Latency} = \max\left(\frac{\mathrm{Ops}}{\mathrm{PEs}},\ \max_i \frac{\mathrm{Access}(L_i)}{BW_i}\right), \tag{16}$$

where PEs denotes the number of effective PEs determined by $T_{s,d,k}$. The term $\mathrm{Access}(L_i)$ represents the total access counts (including fill, read, and write-back) at memory level L_i, and BW_i is its corresponding effective bandwidth as depicted in Figure 2(a).

3.2.3 *Energy Model*

Our energy model accounts for dynamic energy from two primary sources: computation and data movement. The energy for a single layer or node is the sum of these components:

$$\mathrm{Energy} = E_{compute} + E_{datamove}. \tag{17}$$

The total energy of the DNN model is the sum across all layers.

The computation energy is given by:

$$E_{\mathrm{compute}} = \mathrm{Ops} \cdot \mathrm{EnergyPerOp}, \tag{18}$$

where EnergyPerOp is depicted in Figure 2(a).

The data-movement energy is evaluated across the memory hierarchy as the product of the traffic at each interface and its corresponding EPA:

$$E_{datamove} = \sum_i \text{Access}(L_i) \cdot \text{EPA}_i. \tag{19}$$

The final optimization objective is the EDP. Because all components of the model are built differentiably, the gradients of the EDP with respect to the optimization parameters $T_{t,d,m}$, $T_{s,d,m}$, and σ_i can be obtained directly through automatic differentiation.

3.3 Constrained Gradient-Based Optimization

We address this as a constrained optimization problem, minimizing an augmented loss function via gradient descent. This function combines our primary optimization objective (i.e., EDP) with differentiable penalties that encode hardware and logical constraints.

The augmented loss function is defined as:

$$\text{Loss} = \text{EDP} + \lambda\big(P_{\text{map}} + P_{\text{mem}} + P_{\text{align}}\big), \tag{20}$$

where P_{map}, P_{mem}, and P_{align} denote the mapping-validity, memory-capacity, and adjacent-tile alignment constraints, respectively.

3.3.1 Mapping-Validity Penalty. Mapping validity penalty enforces legal temporal and spatial tiling. It penalizes both invalid tiling factors and spatial over-mapping beyond the PE array resource limits. For tiling validity, each tiling factor must be greater than or equal to 1 to avoid ineffective mappings. Therefore, we add a quadratic penalty proportional to the deviation below 1:

$$P_{\text{valid}} = \sum_{d,m} \big(\max(0,\ 1 - T_{d,m})\big)^2. \tag{21}$$

For spatial mapping, the total number of allocated PEs across spatially parallel dimensions should not exceed the available array size N_{PE}. We thus add a quadratic penalty term when this constraint is violated:

$$P_{\text{spatial}} = \big(\max(0, \prod_d T_{s,d,m} - N_{\text{PE}})\big)^2, \tag{22}$$

where m denotes the memory level directly associated with the PE array. The overall invalid mapping penalty combines both terms:

$$P_{\text{map}} = P_{\text{valid}} + P_{\text{spatial}}. \tag{23}$$

3.3.2 Memory-Capacity Penalty. The memory capacity penalty ensures that the storage requirements do not exceed the physical capacity of any on-chip memory buffer. For each fusion group G, we assume that the weights and intermediate activations of all nodes $v \in G$ must reside simultaneously in the buffer at a given memory level i.

The total size requirement for group G at level i, denoted SizeReq(G, i), is computed as the sum of the memory usage for all layers or nodes within that group:

$$\text{SizeReq}(G, i) = \sum_{v \in G} \big(S_W(v, i) + S_I(v, i)\big), \tag{24}$$

where $S_W(v, i)$ and $S_I(v, i)$ are the memory sizes of the resident weights and activations which can be derived using Equation (5), respectively, for node v mapped to memory level i.

A squared penalty is applied when this requirement exceeds the physical capacity of that level, C_i:

$$\text{P}_{\text{mem}}(G, i) = \big(\max(0, \text{SizeReq}(G, i) - C_i)\big)^2. \tag{25}$$

The overall memory capacity penalty, P_{mem}, aggregates these violations across all fusion groups and all relevant on-chip memory levels.

3.3.3 Adjacent-tile Alignment Penalty. Adjacent-tile alignment constraints enforce geometric compatibility between consecutive layers or nodes within each fusion group. For any two adjacent layers v_i and v_{i+1} in a fusion group G, the output tile of v_i must match the input tile of v_{i+1} in shape to guarantee correct data reuse and fusion alignment. Let $\mathbf{o}_{v_i} = (o_p, o_q, o_k)$ denote the output tile shape of layer v_i, and $\mathbf{i}_{v_{i+1}} = (i_h, i_w, i_c)$ the input tile shape of its successor. We define a quadratic alignment penalty as:

$$P_{\text{align}}(G) = \sum_{(v_i, v_{i+1}) \in G} \big\|\mathbf{o}_{v_i} - \mathbf{i}_{v_{i+1}}\big\|_2^2. \tag{26}$$

The total alignment penalty P_{align} is aggregated across all fusion groups.

This penalty-based formulation simultaneously enforces mapping legality, memory feasibility, and shape alignment, guiding the search toward designs that are both performant and realizable. During optimization, the parameters $T_{t,d,m}$, $T_{s,d,m}$, and σ_i are iteratively updated via gradient-based optimization on their continuous relaxations. After convergence, these relaxed parameters are decoded into integer factors and binary fusion decisions to produce the final deployment strategy (see Section 3.1 for the continuous-to-discrete formulation).

The process continues until convergence, yielding the final deployment strategy.

4 Evaluation

In this section, we first describe the experimental setup. Then, the proposed differentiable cost model is validated. Finally, our gradient-based optimization framework, FADiff, is compared with heuristic and learning-based baselines, as well as the current SOTA layer-wise differentiable framework.

4.1 Experimental Setup

All experiments are conducted using Python 3.10 and PyTorch 2.7 on a personal workstation equipped with an Intel® Core™ Ultra 5 125H processor (1.2 GHz, 12 cores) and 32 GB of DDR5 memory. The target hardware is the Gemmini accelerator [10], evaluated under two representative configurations: a *large* 32×32 array with 64 KB L1 and 512 KB L2, and a *small* 16×16 array with 8 KB L1/L2. These two configurations bracket a typical embedded-scale design space for DNN inference and allow us to study how our approach scales under different compute and memory budgets. The evaluation metric of deployment quality is the EDP.

4.2 Validation of Differentiable Cost Models

We first evaluate the cost model's performance on single-layer workloads against Timeloop [17]/Accelergy [18]. The evaluation focused on two key dimensions: numerical accuracy and ranking consistency. For numerical accuracy, the model achieved a 96% prediction accuracy for memory access counts across a diverse set of operators (standard, depthwise, pointwise, large-kernel convolutions, and fully connected layers). For ranking consistency, it demonstrated perfect correlation in latency estimation (Kendall's $\tau = 1.0000, \rho = 1.0000$) and strong correlation in energy estimation (Kendall's $\tau = 0.7804, \rho = 0.9218$). This strong performance at the single-layer level verifies the model's fundamental accuracy and reliability, providing a solid basis for multi-layer validation.

Table 1: EDP Comparison Across Models and Gemmini Configurations.

Model	Large-Gemmini				Small-Gemmini			
	MICRO'23 [8]	BO [15]	GA [16]	**FADiff**	MICRO'23 [8]	BO [15]	GA [16]	**FADiff**
GPT-3 6.7B	1.59×10^{13}	3.67×10^{14}	2.42×10^{14}	$\mathbf{1.15 \times 10^{13}}$	6.14×10^{13}	3.69×10^{15}	2.05×10^{15}	$\mathbf{4.65 \times 10^{13}}$
VGG19	1.16×10^{13}	3.31×10^{14}	2.29×10^{14}	$\mathbf{9.62 \times 10^{12}}$	1.99×10^{13}	8.37×10^{14}	6.41×10^{14}	$\mathbf{1.66 \times 10^{13}}$
VGG16	6.82×10^{12}	1.82×10^{14}	8.65×10^{13}	$\mathbf{6.16 \times 10^{12}}$	9.70×10^{12}	6.44×10^{14}	4.84×10^{14}	$\mathbf{8.33 \times 10^{12}}$
MobileNetV1	2.29×10^{11}	1.60×10^{13}	9.11×10^{12}	$\mathbf{1.67 \times 10^{11}}$	1.44×10^{12}	2.60×10^{13}	2.53×10^{13}	$\mathbf{1.37 \times 10^{12}}$
ResNet18	2.21×10^{10}	4.03×10^{12}	2.98×10^{12}	$\mathbf{2.07 \times 10^{10}}$	2.23×10^{10}	8.13×10^{12}	9.26×10^{12}	$\mathbf{2.13 \times 10^{10}}$
Average	6.91×10^{12}	1.80×10^{14}	1.14×10^{14}	$\mathbf{5.49 \times 10^{12}}$	1.85×10^{13}	1.04×10^{15}	6.42×10^{14}	$\mathbf{1.46 \times 10^{13}}$

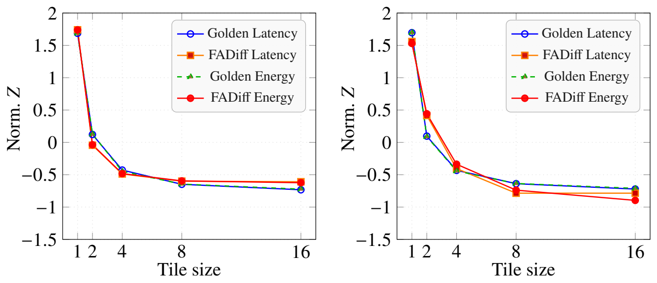

Figure 3: Comparison of Z-score normalized trend. Left: two-layer fusion. Right: three-layer fusion.

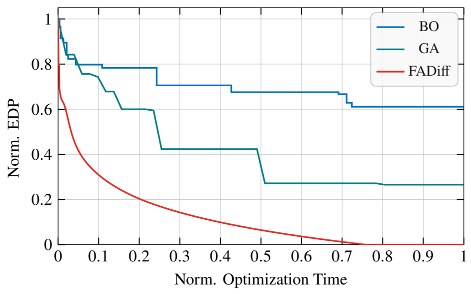

Figure 4: EDP vs. optimization time for GA, BO, and our gradient-based method; with the same time budgets, our method converges faster to lower EDP.

We further evaluate fused operators of multi-layer by comparing against DeFiNES [2], whose analytical model has been calibrated to silicon. Because absolute latency and energy numbers depend on process and microarchitectural details, we again focus on ranking consistency. As shown in Figure 3, the Z-score normalized latency and energy trends of our model closely match DeFiNES for both two- and three-layer fusion. This validates the accuracy of our cost models, providing a robust foundation for gradient-based optimization.

4.3 Comparison of Optimization Quality

4.3.1 Methodological Baselines . We first compare two representative optimization methods to assess the effectiveness of our gradient-based method in this high-dimensional problem, which directly utilizes the differentiable cost model. Specifically, we evaluate Genetic Algorithm [16] as a heuristic baseline, Bayesian Optimization [15] as a learning-based baseline, which are two most commonly used methods in this problem. All experiments use the same search spaces and the large-Gemmini configuration described in Section 4.1, and each method is allocated the same time budget for a fair comparison.

As shown in Figure 4, our gradient-based method reaches substantially lower EDP with far less time compared to GA and BO under the same time budget. This highlights the advantage of leveraging analytical gradients once a differentiable cost model is available, while heuristic and learning-based methods struggle with high-dimensional optimization problems. To connect these optimization dynamics with end-to-end deployment quality, we also run GA and BO on the full model suite and report their best EDP in Table 1 together with the DOSA [8]baseline and our method.

4.3.2 Differentiable Framework Baselines . We further compare FADiff with the layer-wise differentiable framework DOSA [8], which is also gradient-based but optimizes each layer independently, whereas FADiff jointly optimizes layer fusion and mapping. The comparison is performed on the two Gemmini configurations in Section 4.1 under the same time budget. The evaluation suite includes representative convolutional DNNs [19–21] and an LLM workload based on GPT-3-6.7B [22], modeled as a decoder-only Transformer for which we optimize the MHA block [23] and feed-forward network (FFN). The key tensor dimensions of the MHA block are shown in Figure 2(b), and the FFN uses a hidden dimension of 16,384. This diverse set of models, together with the two distinct hardware configurations, is chosen to demonstrate our framework's scalability and generality.

As shown in Table 1, FADiff consistently achieves the lowest EDP across all five workloads on both Gemmini configurations. Compared with the DOSA baseline [8], it reduces EDP by about 18% on the Large-Gemmini and 13% on the Small-Gemmini on average, while GA and BO remain one to two orders of magnitude worse under the same time budget, consistent with the optimization traces in Figure 4 and indicating that heuristic and learning-based methods are ineffective in such high-dimensional spaces.

These results highlight two key trends. First, on all evaluated models and Gemmini configurations, FADiff never degrades the EDP relative to any of the baselines, indicating that our fusion-aware, gradient-based search is robust across both Transformer and convolutional workloads. Second, the benefit of layer fusion is generally larger on the Large-Gemmini than on the Small-Gemmini, especially for GPT-3 6.7B and MobileNetV1, where the larger on-chip buffers make it possible to keep fused activations resident on chip and thereby reduce expensive DRAM accesses. In contrast, on ResNet18 and on the Small-Gemmini configuration, the residual branches and smaller buffer capacity limit the number and size of feasible fusion groups, leading to more modest but still consistently positive improvements.

We attribute these gains to jointly optimizing fusion and mapping in a single differentiable framework and to directly exploiting analytical gradients, which together expose inter-layer optimization opportunities beyond the reach of layer-wise schemes (i.e., DOSA [8]) and heuristic or learning-based search methods.

5 Conclusion

We present FADiff, a unified gradient-based framework that co-optimizes inter-layer fusion and intra-layer mapping via differentiable analytical models and penalty terms. FADiff achieves substantial EDP improvements, demonstrating the promise of differentiable optimization for scalable DNN deployment on tensor accelerators.

References

[1] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*, 2018, pp. 579–594.

[2] L. Mei, K. Goetschalckx, A. Symons, and M. Verhelst, "Defines: Enabling fast exploration of the depth-first scheduling space for dnn accelerators through analytical modeling," in *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 570–583.

[3] J. Cai, X. Wang, M. Gao, S. Peng, Z. Zhu, Y. Wei, Z. Wu, and K. Ma, "Soma: Identifying, exploring, and understanding the dram communication scheduling space for dnn accelerators," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 533–548.

[4] C. Bai, X. Wei, Y. Zhuo, Y. Cai, B. Yu, and Y. Xie, "Klotski: Dnn model orchestration framework for dataflow architecture accelerators," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[5] C. Bai, X. Wei, Y. Zhuo, Y. Cai, H. Zheng, B. Yu, and Y. Xie, "Klotski v2: Improved dnn model orchestration framework for dataflow architecture accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 3, pp. 1045–1057, 2025.

[6] Q. Sun, C. Bai, H. Geng, and B. Yu, "Deep neural network hardware deployment optimization via advanced active learning," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1510–1515.

[7] Q. Sun, C. Bai, T. Chen, H. Geng, X. Zhang, Y. Bai, and B. Yu, "Fast and efficient dnn deployment via deep gaussian transfer learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5380–5390.

[8] C. Hong, Q. Huang, G. Dinh, M. Subedar, and Y. S. Shao, "Dosa: Differentiable model-based one-loop search for dnn accelerators," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 209–224.

[9] Y. Zhao, H. Sharif, V. Adve, and S. Misailovic, "Felix: Optimizing tensor programs with gradient descent," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, p. 367–381.

[10] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 769–774.

[11] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[12] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *arXiv preprint arXiv:1903.05662*, 2019.

[13] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[14] Q. Huang, P.-A. Tsai, J. S. Emer, and A. Parashar, "Mind the gap: Attainable data movement and operational intensity bounds for tensor algorithms," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 150–166.

[15] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[16] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI, USA: University of Michigan Press, 1975.

[17] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[18] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[22] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, no. 3, p. 3, 2020.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.