

# LATIAS: A General Architecture-Operator Model for Spatial Accelerators with Complex Topology and Memory Hierarchy

Chengrui Zhang<sup>1</sup>, Liancheng Jia<sup>1</sup>, Chu Wang<sup>1</sup>, Tianqi Li<sup>1</sup>, Renze Chen<sup>1</sup>, Xiuping Cui<sup>1</sup>,  
Size Zheng<sup>1</sup>, Shengen Yan<sup>2</sup>, Xiuhong Li<sup>1</sup>, Yu Wang<sup>2</sup>, Xiang Chen<sup>1</sup>, and Yun (Eric) Liang<sup>1\*</sup>,

<sup>1</sup>Peking University, Beijing, China

<sup>2</sup>Tsinghua University, Beijing, China

**Abstract**—Spatial accelerators are widely deployed for deep neural networks, but their architectural diversity—from hierarchical to dataflow designs—makes accurate architecture-operator modeling difficult, limiting operator optimization and hardware utilization. Existing models abstract hardware as hierarchical chains and operators as loop trees, which cannot capture essential features of modern dataflow accelerators, including heterogeneous processing elements (PEs), uni-directional interconnects, and cross-PE memory hierarchies, leading to inaccurate latency prediction. We propose LATIAS, a unified framework that introduces (1) an architecture graph with uni-directional edges to represent arbitrary topologies, and (2) a dataflow-aware tile-centric notation that augments loop trees with transfer nodes to model diverse dataflows. Building on these, LATIAS further provides a graph-guided tree analysis that accurately resolves tensor residency and latency under hardware constraints. Experiments on representative operators (GEMM, vector, fused vector) and operator shapes extracted from DNNs (BERT, ViT, T5) on Huawei Ascend 910B3 show that LATIAS achieves over 0.99 correlation with runtime measurements—substantially outperforming prior models—and provides actionable insights for architectural design.

**Index Terms**—Fusion, Accelerator, Simulation and modeling

## I. INTRODUCTION

Numerous spatial accelerators have been developed to support neural network applications [1]–[6], typically built on multiply-accumulate (MAC) units but differing in memory hierarchies, interconnects, and compute capabilities, leading to diverse performance. Structurally, these accelerators fall into two categories (Fig. 1(a)(c)). Hierarchical architectures (e.g., NVIDIA P100 [7]) organize memory and homogeneous PEs into layered structures with parent-child data movement and PEs at the leaves. In contrast, dataflow architectures (e.g., Ascend NPUs [8], Groq [9]) feature heterogeneous PEs (e.g., SIMD, GEMM), uni-directional interconnects for pipelining, and cross-PE memory hierarchies where computation can occur between memory levels (Fig. 1(c)(d)).

Efficient on-chip operator dataflows are essential for improving resource utilization, data reuse, and memory efficiency on such accelerators [10], [11]. Existing approaches fall into two categories: hand-tuning [10], [12], which is expertise-intensive, time-consuming, and specialized, and automated Design Space Exploration (DSE) based on architecture-operator models, where polyhedral-based models [13]–[16] explore tiling with hierarchical architecture abstractions, and [17], [18] exploit operator loop tree properties to minimize data movements.

Fundamentally, these models assume that architectures can be abstracted as a hierarchical bidirectional chain (Fig. 1(b)), where only adjacent levels communicate bidirectionally and PEs are homogeneous. While sufficient for hierarchical accelerators, this abstraction fails on modern designs, which feature heterogeneous PEs, uni-directional interconnects, and cross-PE memory hierarchies. Consequently, chain-based models misrepresent feasible data paths, resulting in **inaccurate bandwidth and latency predictions** and **suboptimal operator optimization**.

Extending architecture-operator modeling beyond hierarchical abstractions requires tackling three challenges: (1) Architecture description. The dataflow accelerators demand a richer abstraction than linear chains: heterogeneous PEs must be modeled with distinct capabilities, uni-directional interconnects constrain feasible communication paths, and cross-PE memory hierarchies break the assumption that computation occurs only at leaves. These factors turn the architecture description from a simple hierarchy into a general graph problem. (2) Operator description. On graph-structured architecture, operators may exhibit tree-based, ring-based, or hybrid transfer patterns (Fig. 2), beyond simple parent-child communication. The challenge is to design a representation enough to capture such diverse dataflows while remaining analyzable. (3) Operator latency analysis. Graph edges create multiple producer-consumer paths and allow tensors to bypass ancestors, breaking the assumptions of depth-first-search-based (DFS-based) operator tree analysis [13], [16](Fig. 3). This leads to ambiguity in path selection and risks of misestimating bandwidth and latency, which makes accurate analysis difficult.

To address these challenges, we propose LATIAS—a **unified framework for architecture-operator modeling across diverse accelerators**. The key insight is that a graph-based representation can faithfully capture complex architectural characteristics, enabling precise operator latency analysis and overcoming the limitations of chain-based models.

We introduce an architecture graph that generalizes architecture descriptions beyond hierarchical hierarchies: nodes denote architectural levels (e.g., memory or PEs), and uni-directional edges encode data paths, enabling the modeling of **arbitrary topologies**. To complement this, we propose a dataflow-aware tile-centric notation (DTN), which enriches the operator loop tree with transfer nodes to faithfully capture architecture-induced dataflows. Unlike prior abstractions restricted to parent-child propagation, DTN can naturally represent **diverse data transfer patterns**. By tightly coupling the architecture graph with

Email: zhangchr@stu.pku.edu.cn

\*Corresponding author. Email: ericyun@pku.edu.cn

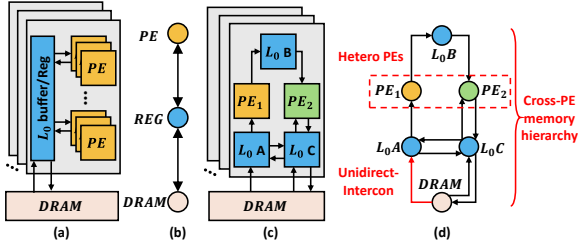


Fig. 1: Architectures and abstractions. (a)(b) Hierarchical. (c)(d) Dataflow.

DTN, our framework delivers the first unified representation of hardware and operators that scales across modern accelerators, providing a solid foundation for accurate and generalizable operator analysis.

Building on this representation, we develop a graph-guided tree analysis framework, which extends DFS traversal on the DTN-based operator tree by explicitly incorporating architecture-graph connections and transfer nodes. Unlike conventional DFS that assumes strict parent-child propagation—and thus misrepresents data movement when shortcuts or cross-level paths exist—our method dynamically resolves **feasible producer-consumer paths under graph constraints**, ensuring that tensor propagation faithfully matches the actual dataflows of modern accelerators. Within this framework, we introduce two analysis passes: (1) tensor residency, which systematically determines the input/output tensors of each loop-tree node, and (2) performance evaluation, which quantifies data movement and latency across nodes and aggregates them into overall operator latency. Together, these passes provide a robust and accurate foundation for operator modeling across diverse architectures. We also make LATIAS open-source in Github (<https://github.com/pku-liang/LATIAS>).

In summary, our contributions are as follows:

- We propose an architecture graph abstraction for general spatial accelerators, together with a dataflow-aware tile-centric notation for representing operator behaviors.
- We propose a graph-guided tree analysis framework that generalizes conventional DFS by incorporating architecture-graph connections and transfer nodes, enabling operator loop-tree analysis under arbitrary dataflow constraints.
- We validate the proposed model on a real-world spatial accelerator with classic operators, and further demonstrate its robustness under varying resource constraints.

Experiments on Huawei Ascend 910B3 show over 0.99 correlation with real execution across GEMM, vector, and fused vector operators from the transformer, confirming strong trend consistency. Our method further demonstrates robustness across diverse tiling and resources configurations, while providing valuable guidance for architectural design. In addition, the implementation of LATIAS will be open-source in the future.

## II. BACKGROUND

### A. Dataflow Analysis Models

Designing efficient operator dataflows is challenging, and a range of operator performance models have been proposed.

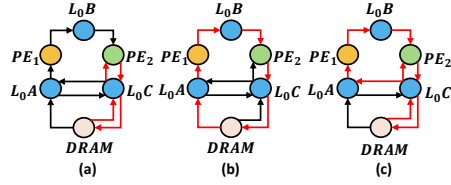


Fig. 2: Different operator data transfer patterns. (red line). (a) Tree. (b) Ring. (c) Hybrid.

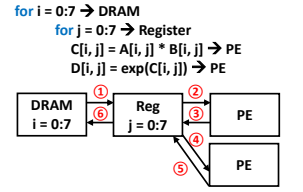


Fig. 3: Operator tree representation & analysis.

Early frameworks such as Timeloop [13], MAESTRO [14], and TENET [15] focus on single operators with perfectly nested loop structures, providing detailed modeling of tiling and mapping. However, they are limited when facing fusion dataflows, where one operator’s iteration space is embedded into another, yielding non-perfect loops that cannot be represented within their abstractions. To address this gap, HASCO [18] and H2H [17] approximate fusion by separately evaluating operators and pruning redundant transfers, while more recent works such as SET [19] and TileFlow [16] extend tile-centric notations and tree-based analyses to model fusion, capturing compute ordering, resource binding, and multi-dimensional tiling.

Across these models, two types of tile relationships are commonly defined. (1) Intra-tile relationships describe execution within a tile: *spatial* (parallel loop execution across units) and *temporal* (sequential loop execution on the same unit). (2) Inter-tile relationships capture interactions among tiles, including *sequential* execution, *sharing* of input tensors, *parallel* execution across resources, and *pipeline* execution where the output of one tile feeds into the next. These abstractions form the analytical basis of prior work and will be extended in our model to handle more general dataflow architectures.

### B. Architecture, Operator, and Analysis in Existing Models

Hardware description. Existing models typically abstract hierarchical accelerators as linear chains (Fig. 1(b)), since this provides a simple and analyzable mapping between memory and PE levels. In such chains, memory and PEs are connected bidirectionally, and PEs are confined to the leaf nodes, which implicitly assumes homogeneous PEs and symmetric interconnects. However, dataflow architectures break these assumptions in three key ways. First, modeling heterogeneous PEs would require multiple arithmetic nodes, which exceeds the expressive capacity of a single-node chain abstraction. Second, chain abstractions constrain data movement to adjacent parent-child levels, making it impossible to represent direct cross-level communication. Third, dataflow accelerators often interleave PEs with memory levels, forming cross-PE hierarchies. Such designs cannot be captured by chain-based abstractions, which strictly enforce PEs to appear only at the terminal node.

Operator description. Operators are generally represented as loop trees (Fig. 3), because loop nests naturally map to recursive parent-child structures. This representation is sufficient for tree-structured dataflows (Fig. 2(a)), in which data moves hierarchically and bidirectionally. However, it fails to capture more complex cases. In ring-based transfers (Fig. 2(b)), data circulates

across multiple nodes and propagates sequentially downstream. In hybrid cases (Fig. 2(c)), tree-based and ring-based transfers are coupled at certain nodes, causing different parts of the path to follow different rules (e.g., sequential propagation or parent–child write-back). Such diverse dataflows cannot be expressed within the constraints of loop-tree representations.

Performance analysis. Latency is typically estimated by traversing the operator loop tree in depth-first order, because DFS matches the natural execution order of nested loops and allows recursive aggregation of data movement (①–⑥ in Fig. 3). However, this process assumes strict parent–child propagation of tensors. Once architecture-defined shortcuts or multiple producer–consumer paths are introduced, this assumption no longer holds. As a result, DFS-based analyses either insert spurious intermediate nodes into the data path or misestimate bandwidth and latency, leading to inaccurate performance evaluations.

### III. ARCHITECTURE AND OPERATOR DESCRIPTIONS

In this section, we describe the architecture and operator descriptions in Sec. III-A and Sec. III-B, respectively.

#### A. Architecture description

We represent the architecture as a directed graph  $G = (\text{Nodes}, \text{Edges})$  composed of two fundamental elements.

- Nodes are categorized into *MemUnits* (memory components) and *ArithUnits* (processing elements). Each node maintains type-specific attributes. For example, a RegFile node records (1) basic properties such as name and degree of parallelism, (2) memory features such as class and capacity, and (3) other performance-related specifications. This fine-grained representation enables precise characterization of architectural components.
- Edges are **uni-directional** connections between nodes, annotated with properties such as bandwidth and inter-connection type (broadcast, unicast, full-/half-duplex).

Unlike chain-based abstractions, this graph representation decouples read and write paths, offering greater flexibility to capture heterogeneous PEs, uni-directional interconnections, and cross-PE memory hierarchies (Fig. 4(b)).

#### B. Dataflow-aware operator description

Given an operator loop tree  $T$  (Fig. 4(a)) and an architecture graph  $G$  (Fig. 4(b)), we define an **overlay** procedure that integrates them into a unified representation  $\tilde{T} = (T, G)$  (Fig. 4(c)). During overlay, structural constraints from the architecture graph are imposed on the operator tree. Specifically, when a tensor produced at node  $u \in \tilde{T}$  is transferred to node  $v \in \tilde{T}$ , the overlay step identifies the corresponding architecture nodes in  $G$  and augments the tree with explicit *transmit tiles* representing the architecture-defined path between  $u$  and  $v$ .

We denote this augmented structure as the **dataflow-aware tile-centric notation**, which provides a **three-stage operator representation**:

- Specs: including tile type (TT) and dataflow type (DT).
- Tiling factors: partitioning strategies at memory or arithmetic levels.

- Child nodes: parent-children dependency.

Formally, a node tile at tree level  $n$  is defined as:

$$\tilde{T}_n = \underbrace{\{\text{DT}, \text{TT}\}}_{\text{Specs}} \underbrace{[l_n^1, l_n^2, \dots]}_{\text{Tiling factors}} \underbrace{(\tilde{T}_{n-1}^1, \tilde{T}_{n-1}^2, \dots)}_{\text{Child nodes}}.$$

Here,

- **Memory/Arithmetic tiles**: Represent tiling at memory or arithmetic levels. Memory tiles allow flexible tiling factors, while arithmetic tiles are fixed by the architecture.
- **Scope tiles**: Capture inter-tile relationships (sequential, sharing, parallel, pipeline).
- **Transmit tiles**: Bridge the operator tree and architecture graph by modeling explicit data transfers. A transmit tile specifies that a tensor generated by a leaf node is delivered to a target architecture level. For example, in Fig. 4, the transmit tile in (c) corresponds to the red-arrow connection in (b). Their presence enables arbitrary dataflows to be expressed within a tree and allows clear differentiation between direct parent–child propagation (*DT: Write-Back*) and architecture-guided transfers (*DT: DataFlow*).

Through this overlay process, operators can be systematically represented as DTNs, where each node conforms to a pre-defined tile type. This unification of operator and architecture descriptions lays the foundation for subsequent residency and performance analysis.

### IV. GRAPH-GUIDED TREE ANALYSIS

Since the aforementioned description breaks the default parent–child propagation rule in DFS-based tree analysis under the dataflow architecture, in this section, we propose a graph-guided tree analysis framework to adapt the operator analysis to the architecture graph and execute performance evaluation. Our analysis framework first analyzes tensor residency across the architecture graph and then adapt DFS traversal to follow architecture-constrained paths for performance evaluation.

#### A. Tensor Residency Analysis

We define two primitives to model tensor propagation in the overlay tree.

- **Route & Meet**: Given a producer node  $p$  and a consumer node  $c$ , the function  $\text{Route}(p, c | \tilde{T})$  identifies their feasible architecture paths under graph constraints (Red arrow in Fig. 6), and  $\text{Meet}(p, c)$  denotes their lowest common node. The Route function operates under three distinct patterns: (1) **Write-Back Pattern**. In this case, no transmit node exists in the tree. The tensor produced by the producer is propagated upward along the producer’s ancestors, while the consumer’s demand is also traced upward. Their common node is then determined using the Meet function (e.g., node 1 in Fig. 6(a)). (2) **Dataflow Pattern I**. Here, transmit nodes are explicitly present in the tree. The first case occurs when the producer resides at the same hierarchical level as the next transmit node. In this scenario, the tensor produced by the producer is directly forwarded to the target node specified by the transmit node (e.g., node 3 in Fig. 6(b)).

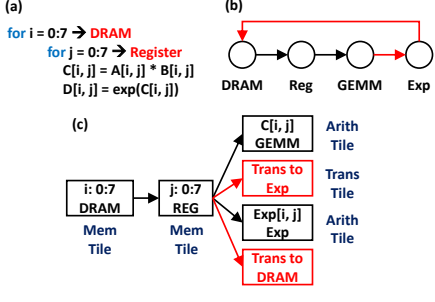


Fig. 4: Example of the overlay process. (a) Operator loop tree  $T$ . (b) Architecture graph  $G$ . (c) Overlay tree ( $\tilde{T} = (T, G)$ ).

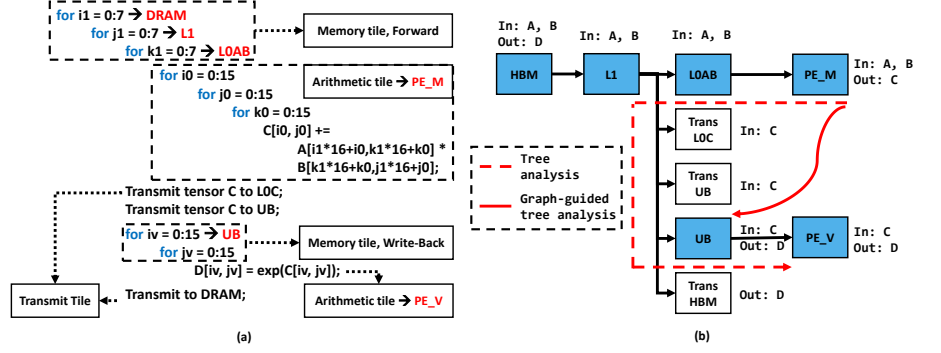


Fig. 5: Operator description. (a) Operator loop description. (b) Operator DTN description.

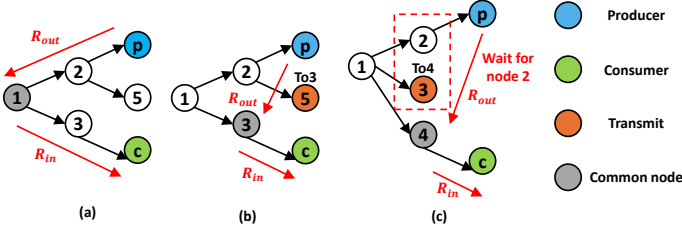


Fig. 6: Route patterns. (a) Write-back. (b) Dataflow (Transfer under the same levels). (c) Transfer under different levels.

(3) Dataflow Pattern II. In this case, the transmit node is located above the producer in the hierarchy. This indicates that data transmission is deferred until computations at the peer nodes of the transmit node are completed. For example, in Fig. 6(c), the tensor generated by  $p$  waits for node 2 to finish before transmission occurs.

- Residency Annotation: Each node  $n$  maintains two sets:  $\rho_{in}(n)$  and  $\rho_{out}(n)$ , denoting its input and output tensors. For a tensor produced at  $p$  and consumed at  $c$ , we annotate all nodes along  $Route(p, c)$  with this tensor:

$$\rho_{out}(p) \supseteq \{t\}, \quad \rho_{in}(c) \supseteq \{t\}, \quad (1)$$

$$\forall m \in Route(p \rightarrow c) : \rho_{in}(m) \supseteq \{t\}.$$

- Dataflow Mode: By comparing producer and consumer paths, we classify the transfer as either (1) *Write-Back* (adjacent nodes) or (2) *DataFlow* (requiring transmit tiles).

Fig. 5 illustrates a fusion example of GEMM+EXP, in which DRAM, L1, L0AB, L0C, and UB are memory units, and PE\_M, PE\_V are matrix/vector PEs. Here, results from PE\_M are directly sent to both L0C and UB for fusion with vector PE, while previous tree analysis (red dashed line) would wrongly assume propagation upward through L0AB and L1 before reaching UB—paths absent in real architecture. Under our residency analysis, we can correctly annotate the input and output tensors as shown in Fig. 5(b) (red line).

## B. Performance evaluation

On top of the overlay tree  $\tilde{T}$ , performance evaluation is formulated in terms of two fundamental primitives and a set of composition laws.

- Data Movement Estimation (Slice Algebra). Each node of  $\tilde{T}$  holds input/output tensor lists which are acquired by

the residency analysis. Each tensor stored at node  $\tilde{T}_n$  is represented as a slice at time  $t$ , with  $b_i^t$  and  $e_i^t$  denoting the beginning and ending addresses of dimension  $i$ .

$$Slice(\tilde{T}_n, t) : Tensor_{Tile}^t [b_0^t : e_0^t, b_1^t : e_1^t, \dots, b_{D-1}^t : e_{D-1}^t] \quad (2)$$

The temporal data movement between consecutive time steps is defined as the set difference (updating the tensor):

$$DM(\tilde{T}_n, t) = \|Slice(\tilde{T}_n, t) \setminus Slice(\tilde{T}_n, t-1)\|. \quad (3)$$

- Node Latency Decomposition. The latency of a tile node is decomposed into three components:

$$Lat(\tilde{T}_n) = L_{load}(\tilde{T}_n) + L_{pro}(\tilde{T}_n) + L_{store}(\tilde{T}_n). \quad (4)$$

Here,  $L_{load}$  and  $L_{store}$  are derived from the transfer volume and available bandwidth, while  $L_{pro}$  depends recursively on the latency of sub-tiles. Here we assume full pipelining and double buffering, which means that different latencies from different tiles can be overlapped.

- Latency Composition Laws. Depending on intra-/inter-tile execution modes, child latencies are combined using the formulations in Table I. Here intra-tile analysis distinguishes spatial and temporal execution, and we suppose  $\tilde{T}_n$  contains  $M$  local tiles denoted as  $\tilde{T}_{lc}^m$ , and the children of the local tiles are denoted as  $\tilde{T}_{chd}^m$ . Inter-tile analysis considers sequential, sharing, parallel, and pipeline modes, and we suppose  $\tilde{T}_n$  contains  $K$  sub-tiles and  $R_n$  denotes the tiling factor of  $\tilde{T}_n$ . **Interconnection types** are also considered, mainly affecting input/output latency.

Overall, the performance analysis procedure is outlined in Alg. 1. Each type of node invokes a different function to compute its data movement and latency. For arithmetic and transmit nodes, **tensor update** of current node is identified directly and the corresponding data movements are calculated, with latency derived from the **source-destination bandwidth**. For scope nodes, the analysis iteratively visits their children nodes and aggregates the results according to the inter-node execution mode specified by the scope. For memory nodes, which contain tiling information, the analysis traverses each loop dimension and computes the overall latency while accounting for the intra-node execution mode. The **TensorList** in this algorithm comes from the previous tensor residency analysis.

### Algorithm 1: Performance Analysis

```

1 Function PerformancePass (Node)
2   if IsArithOrTransNode (Node) then
3     IOLat, DM  $\leftarrow$  UpdateTensor (Node.TensorList);
4     Lat = IOLat + (ComputeLat);
5   if IsScopeNode (Node) then
6     Lat.clear();
7     foreach Child  $\in$  children(Node) do
8       PerformancePass (Child);
9       Lat.push(Child.Lat);
10    Node.Lat  $\leftarrow$  CallLat (Lat, InterMode);
11  if IsMemoryNode (Node) then
12    VisitLoop (Node);
13 Function VisitLoop (Node)
14  Lat.clear();
15  IOLat, DM  $\leftarrow$  UpdateTensor (Node.TensorList);
16  for loop  $\in$  Node.tilingFactor do
17    InterLat.clear();
18    foreach Child  $\in$  Children(Node) do
19      PerformancePass (Child);
20      InterLat.push(Child.Lat);
21  Node.Lat  $\leftarrow$  CallLat (InterLat, IntraMode);

```

TABLE I: Analysis formulations.

Intra-tile Latency Analysis	
Spatial & unicast	$\max_m \{L_{T_{1c}}^m, S_{T_{1c}}^m\} \times M + \max_n \text{Lat}_{T_{chd}}^n$
Spatial & broadcast	$\max_n \text{Lat}_{T_{1c}}^n$
Temporal	$\max_m \{L_{T_{1c}}^m, S_{T_{1c}}^m, \text{Lat}_{T_{chd}}^m\} * M$
Inter-tile Latency Analysis	
Perfect loop	MAESTRO( $T_n, \text{Spec}_n$ ) [13]–[15]
Seq or shar	$\max\{\text{Lat}_{DM}, \sum_k \text{Lat}_{T_{n-1}}^k\} * R_n$
Paral	$\max\{\text{Lat}_{DM}, \max_k \text{Lat}_{T_{n-1}}^k\} * R_n$
Pipe	$\max\{\text{Lat}_{DM}, \text{pipe}(T_{n-1})\} * R_n$
Auxiliary Function For Latency Analysis	
Pipeline calculation	$\text{pipe}(T_{n-1}) = K * \max\{L_{T_{n-1}}, \text{Lat}_{T_{n-2}}, S_{T_{n-1}}\}$
Full-duplex latency	$\text{Lat}_{DM} = \text{Max}\{L_{T_n}, S_{T_n}\}$
Half-duplex latency	$\text{Lat}_{DM} = L_{T_n} + S_{T_n}$

## V. EVALUATION

### A. Evaluation Setup

**Platforms.** Our evaluation centers on the state-of-the-art spatial dataflow accelerator Huawei Ascend 910B3 (DaVinci architecture [20]). Platform specifications are shown in Fig. 7. The Cube Core (left) is specialized for GEMM computations and follows a **hybrid data transfer path (dataflow architecture)**, while the Vector Core (right) is designed for SIMD computations and follows a **tree-based data transfer path (hierarchical architecture)**. Since the chip naturally incorporates both two architectures, it provides a comprehensive platform to validate the generality and accuracy of our model. Bandwidth specifications, obtained through custom micro-benchmarks, are summarized on the right. For this spatial accelerator platform, we utilize the CANN Software Development Kit provided by Huawei. Baseline performance is measured using ACLNN [21], a cutting-edge official library optimized for accelerating neural network applications. Performance analysis is conducted using the official cycle-level profiling tool – msprof.

**Benchmarking.** Our evaluation dataset consists of three representative operators extracted from the Transformer [22]: a standalone GEMM operator, a standalone vector operator (VEC:

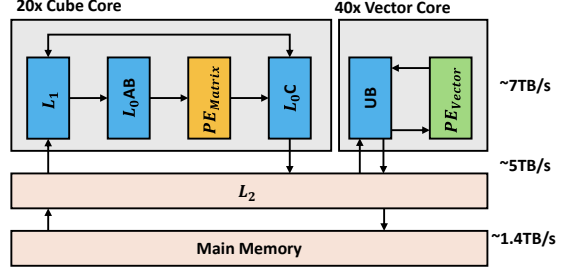


Fig. 7: Huawei Ascend 910B3.

TABLE II: Benchmark of verification operator shape.

operator	M	N	K	Num
GEMM	[128-8192]	[128-8192]	[128-8192]	1200
VEC	[128-4096]	[128-4096]	-	1024
FUS	[128-4096]	[128-4096]	-	1024

TABLE III: Benchmark of popular transformer operator shape.

Network	GEMM-(M, N, K)	VEC-(M,N)
Bert-S/B/L	[512, (512/768/1024), (512/768/1024)]	[512, (2048/3072/4096)]
ViT14-B/L/H	[256, (768,1024,1280), (768,1024,1280)]	[256, (3072,4096,5120)]
T5	[1024, 1024, 1024]	[1024, 4096]

SUB), and a fused vector operator (FUS: SUB+EXP). On the target accelerator, the GEMM operator exhibits a hybrid data transfer pattern, whereas the VEC and FUS operators follow a tree-based pattern. The detailed operator configurations are summarized in Table II. In addition, we further evaluate our model on full neural networks, including BERT [23], ViT [24], and T5 [25]. Each network is tested under three architecture utilization scenarios to validate the robustness of our model: (1) HP: using half of the PEs without tiling at inner memory levels; (2) AP: using all PEs without tiling at inner memory levels; (3) AP-tile (Ours): using all PEs with tiling enabled.

**Comparison.** We compare our approach against the state-of-the-art baseline TileFlow [16].

### B. Model Accuracy

Fig. 8 compares the latency predicted by our model, TileFlow and the measured latency of ACLNN on the benchmark. The comparison is shown in Table IV. To quantify the model accuracy, we adopt the Pearson Correlation Coefficient  $\tau$  [26] and the Mean Absolute Percentage Error MAPE [27] compared to the real ACLNN results. We further define overall model accuracy as  $\text{Acc} = \frac{\tau}{\text{MAPE}}$ , which balances the strength of correlation with the relative prediction error, thereby providing a single metric that reflects both consistency and precision.

Our results demonstrate **strong trend-following capability**: for GEMM, VEC, and FUS operators, our model achieves correlation coefficients of 0.996, 0.998, and 0.997 against ACLNN, respectively. By contrast, TileFlow cannot model GEMM on this dataflow-based accelerator, and even for VEC and FUS its correlations drop to 0.658 and 0.650. The instability of TileFlow primarily stems from its reliance on perfectly nested tiling (e.g.,  $12 = 3 \times 2 \times 2$ ). For irregular shapes (e.g.,  $11 = 11 \times 1$ ), TileFlow struggles to adjust tiling factors, leading to significant prediction errors. In contrast, our model naturally **supports imperfect tiling** (e.g.,  $11 = 3 \times 4(3) = 2 * 4 + 3$ , where

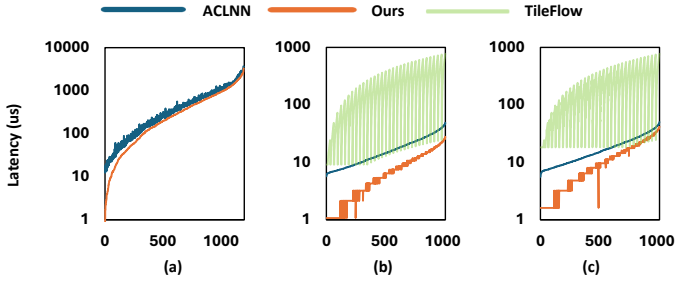


Fig. 8: Simulation results compared with ACLNN and TileFlow. (a) GEMM. (b) VEC. (c) FUS.

TABLE IV: Comparison with TileFlow.

Operator	TileFlow		Ours			
	Corr	MAPE	Acc	Corr	MAPE	Acc
GEMM	-	-	-	<b>0.996</b>	<b>0.261</b>	<b>3.82</b>
VEC	0.659	10.342	0.0638	<b>0.998</b>	<b>0.614</b>	<b>1.63</b>
FUS	0.65	10.176	0.064	<b>0.997</b>	<b>0.445</b>	<b>2.24</b>

the last iteration uses 3 instead of 4), thus maintaining high accuracy even on non-divisible dimensions.

In addition, Fig. 8 shows that our model exhibits larger percentage deviations from ACLNN at lower latencies (typically corresponding to smaller input shapes), whereas the predictions converge more closely at higher latencies (larger shapes). This effect arises from the gap between theoretical and effective bandwidth [28]: when the data volume is small, the effective bandwidth is significantly lower than the theoretical peak, while our model assumes the latter. As the data volume increases, effective bandwidth approaches the theoretical value, leading to better agreement with real results.

### C. Robustness and Architecture Sensitivity Analysis

We evaluate both the robustness of our model and the impact of architecture modifications on operator performance, aiming to provide insights for architecture design. The modeling results for the operators listed in Table III are shown in Fig. 9. Subfigures (a) and (b) present the latency and data movement analysis for the GEMM operator, while (c) and (d) illustrate the latency results for the VEC and FUS operators, respectively.

For the GEMM operator (Fig. 9(a)(b)), the average latency under the HP configuration is 1.96 $\times$  that of the AP configuration, consistent with the expected slowdown caused by halving the number of processing elements. In contrast, the data movement remains identical across HP and AP, since the number of cores allocated does not influence the total data volume transferred.

When comparing AP and AP-tile, we observe that tiling reduces latency. By increasing the amount of data transferred per instruction, tiling improves bandwidth utilization and enhances pipelining. In GEMM computations, tiling further enables partial reuse of input data, which decreases overall data movement (Fig. 9(b)) and consequently reduces total latency (Fig. 9(a)).

Beyond operator-level performance, our framework also provides guidance for architecture design. Taking GEMM as an example, we analyze the impact of modifying the capacity of L0AB in Fig 7: doubling and halving its original size, denoted as **DOUBLE** and **HALF**, respectively. The corresponding latency and data movement predicted by our model are shown in Fig. 10.

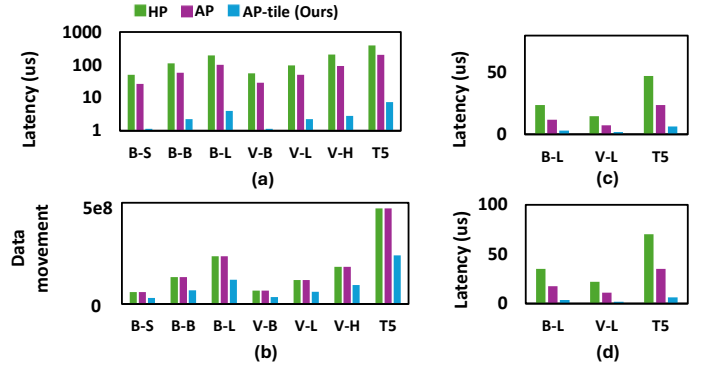


Fig. 9: Robustness analysis. (a)(b) Latency and data movement of GEMM. (c) Latency of VEC. (d) Latency of FUS.

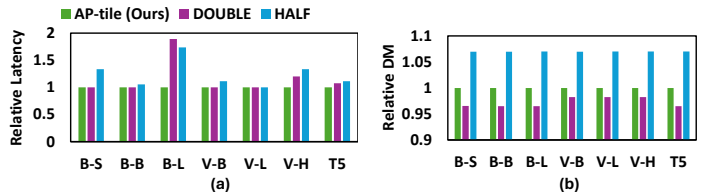


Fig. 10: Architecture sensitivity analysis. (a) Relative Latency. (b) Relative data movement.

As illustrated, the **DOUBLE** configuration achieves an average 2.7% reduction in total data movement compared to the baseline, while **HALF** results in an average increase of 6.9%. However, this reduction in data movement does not always yield performance gains, as observed in the BERT-L, ViT14-L, and T5 cases in Fig. 10(a). Such discrepancies arise from multiple factors, including pipeline bubbles and tensor shape alignment. These results highlight that **simply enlarging or shrinking L0AB memory capacity** does not guarantee performance improvements for GEMM, providing the insight of architecture modification in spatial accelerators.

## VI. CONCLUSION

We presented LATIAS, a unified framework for modeling operator dataflows on spatial accelerators with complex topologies and memory hierarchies. By introducing an architecture graph to capture heterogeneous processing elements, uni-directional interconnects, and cross-PE memory hierarchies, together with a dataflow-aware tile-centric notation to represent diverse dataflows, LATIAS overcomes the limitations of chain-based abstractions. Besides, our graph-guided tree analysis provides accurate tensor residency and latency estimation. Experiments on Huawei Ascend 910B3 with representative operators and operator shapes extracted from full DNNs demonstrate that LATIAS achieves over 0.99 correlation with runtime results and offers practical insights for architecture design.

## ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their insightful suggestions. This work was supported in part by the National Science Foundation of China (NSFC) under Grant No.T2325001.

## REFERENCES

- [1] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.
- [2] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. *ACM SIGARCH Computer Architecture News*, 45(2):535–547, 2017.
- [3] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 977–991, 2021.
- [4] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.
- [5] Liancheng Jia, Zizhang Luo, Liqiang Lu, and Yun Liang. Tensorlib: A spatial accelerator generation framework for tensor algebra. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 865–870, 2021.
- [6] Liancheng Jia, Yuyue Wang, Jingwen Leng, and Yun Liang. Ems: efficient memory subsystem synthesis for spatial accelerators. In *Proceedings of the 59th ACM/IEEE Design Automation Conference, DAC '22*, page 67–72, New York, NY, USA, 2022. Association for Computing Machinery.
- [7] Nvidia. Nvidia pascal architecture whitebook, 2016.
- [8] 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 *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801. IEEE, 2021.
- [9] Dennis Abts, John Kim, Garrin Kimmell, Matthew Boyd, Kris Kang, Sahil Parmar, Andrew Ling, Andrew Bitar, Ibrahim Ahmed, and Jonathan Ross. The groq software-defined scale-out tensor streaming multiprocessor: From chips-to-systems architectural overview. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–69. IEEE Computer Society, 2022.
- [10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
- [11] 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 *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1113–1126. IEEE, 2023.
- [12] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.
- [13] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Bruce Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.
- [14] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE micro*, 40(3):20–29, 2020.
- [15] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. Tenet: A framework for modeling tensor dataflow based on relation-centric notation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 720–733. IEEE, 2021.
- [16] 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 *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1271–1288, 2023.
- [17] Xinyi Zhang, Cong Hao, Peipei Zhou, Alex Jones, and Jingtong Hu. H2h: heterogeneous model to heterogeneous system mapping with computation and communication awareness. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 601–606, 2022.
- [18] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. Hasco: Towards agile hardware and software co-design for tensor computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1055–1068. IEEE, 2021.
- [19] Jingwei Cai, Yuchen Wei, Zuocong Wu, Sen Peng, and Kaisheng Ma. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–17, 2023.
- [20] Huawei. Huawei hardware architecture, 2024.
- [21] Huawei. Huawei aclnn operator library, 2024.
- [22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
- [23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [24] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.
- [25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
- [26] Philip Sedgewick. Pearson’s correlation coefficient. *Bmj*, 345, 2012.
- [27] Wiki. Mean absolute percentage error.
- [28] Ashwin M Aji, James Dinan, Darius Buntinas, Pavan Balaji, Wu-chun Feng, Keith R Bisset, and Rajeev Thakur. Mpi-acc: An integrated and extensible approach to data movement in accelerator-based systems. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 647–654. IEEE, 2012.