

FUSE: A Fragmentation-Free Spatial Multi-Tenant DNN Accelerator via Edge-Universal Graph

1st Zhihao Yang

*Intelligent Software Research Center,
Institute of Software, Chinese Academy of Sciences
Beijing, China
University of Chinese Academy of Sciences
Beijing, China
yangzhihao2021@iscas.ac.cn*

2nd Yongwei Zhao

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
zhaoyongwei@ict.ac.cn*

3rd Tianrui Ma

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
matianrui@ict.ac.cn*

4th Yifan Hao

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
haoyifan@ict.ac.cn*

5th Zhe Fan

*Cambricon Technologies
Beijing, China
fanzhe@cambricon.com*

6th Rui Zhang

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
zhangrui@ict.ac.cn*

7th Zidong Du

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
duzidong@ict.ac.cn*

8th Ling Li

*Intelligent Software
Research Center,
Institute of Software,
Chinese Academy of Sciences
Beijing, China
liling@iscas.ac.cn*

9th Tianshi Chen

*Cambricon Technologies
Beijing, China
tchen@cambricon.com*

10th Yunji Chen

*State Key Lab of Processors,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
cyj@ict.ac.cn*

Abstract—Multi-tenancy is critical for meeting the latency requirements of DNN models concurrently executing on a shared accelerator while improving hardware utilization. Existing accelerators achieve spatial multi-tenancy by dynamically co-locating tasks across tiles. However, to limit hardware cost, such co-location typically relies on constrained interconnects. As a result, data forwarding between some tiles can block forwarding between others, leaving many tiles underutilized—i.e., fragmentation.

In this paper, we propose FUSE, a fragmentation-free spatial multi-tenant DNN accelerator that supports universal, non-blocking tile-to-tile data forwarding at negligible area overhead. Our key insight is that inter-tile data forwarding patterns are naturally acyclic, and can be modeled as Directed Acyclic Graphs (DAGs) instead of dense, bidirectional interconnects in conventional designs. Therefore, the data forwarding network can be built upon Valiant’s Edge-Universal Graph (EUG), a graph construction that efficiently supports DAG embeddings. However, direct hardware implementation of EUG has historically been impractical due to its unbounded critical-path delays. We address this issue with a novel *Temporal-Zone Timing* method that bounds critical paths while preserving functional correctness. Built

on this foundation, FUSE achieves efficient, fragmentation-free multi-tenancy. Post-layout evaluation shows that FUSE improves Quality-of-Service by 1.38 $\times$, throughput by 1.11 $\times$, and lowers area overhead from 19.33% to 6.44%, compared to the state-of-the-art accelerators.

Index Terms—Spatial DNN Task Co-Location, Multi-Tenant DNN Acceleration.

I. INTRODUCTION

Modern human-computer interaction applications, such as Augmented and Virtual Reality (AR/VR), demand multiple real-time Deep Neural Network (DNN) tasks to run concurrently (e.g., object detection, speech recognition). These tasks are typically triggered by multiple low-power sensors on the device (e.g., AR glasses) but executed on a shared edge server (e.g., host NPU) due to power constraints. This execution model calls for *multi-tenant accelerators* that can serve diverse tasks with strict latency requirements. Such accelerators achieve concurrency by partitioning the chip into tiles and dynamically mapping subsets of tiles to different tasks. Through this, spatial multi-tenancy enables concurrent execution of multiple small DNN tasks, improving resource utilization and ensuring Quality of Service (QoS).

Efficient tile-to-tile data forwarding is critical to minimize costly memory access during inter-tile communication.

This work is partially supported by the NSF of China (Grants No.92364202, 62525203, 62341411, 62302478), the National Key R&D Program of China (Grant No.2024YFB4505501), Strategic Priority Research Program of the Chinese Academy of Sciences (Grants No.XDB0660200, XDB0660201, XDB0660202), CAS Project for Young Scientists in Basic Research (YSBR-029) and Youth Innovation Promotion Association CAS.

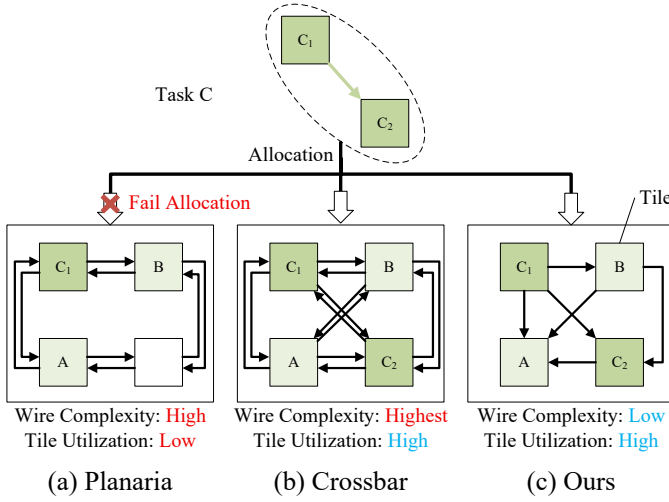


Fig. 1. (a) Ring topology: Task C cannot be fully allocated (i.e., fragmentation). (b) Crossbar topology: Task C can be fully allocated but incurs high area overhead. (c) Ours: Task C can be fully allocated with significantly lower area overhead.

However, achieving universal, non-blocking forwarding among tiles has traditionally required complex interconnects, which impose prohibitive area overhead. Prior solutions (e.g., Planaria [1] and SARA [2]) try to reduce this cost by restricting forwarding to adjacent tiles with constrained topologies, but they critically introduce *fragmentation* during dynamic task mapping: **when tasks require communication between tiles that are not directly connected by the interconnect, some tiles remain underutilized.**

As shown in Fig. 1, consider a task C consisting of two subtasks, C_1 and C_2 , mapped to two tiles with data forwarding required from C_1 to C_2 . When attempting to allocate task C while tasks A and B are already running, Planaria [1] (Fig. 1(a)) can only allocate C_1 due to the absence of a direct data-forwarding path between the two tiles, causing fragmentation (i.e., underutilization of tiles). Using a crossbar interconnect topology (Fig. 1(b)) would enable direct data forwarding between the tiles but introduce extremely high wiring complexity.

To solve this issue, we propose a novel interconnect topology, as shown in Fig. 1(c), which requires minimal wirings to realize nearly-full tile utilization in task mapping. Our theoretical analysis shows that forwarding patterns in spatial multi-tenant DNN accelerators can be represented as bounded-degree Directed Acyclic Graphs (DAGs), thus dense, bidirectional interconnects are unnecessary. This insight allows us to embed forwarding patterns into Valiant’s Edge-Universal Graph (EUG) [3], proposed in 1976, which guarantees universal connectivity with asymptotically optimal circuit complexity—specifically, universal forwarding among four tiles can be realized with only three switches in an EUG-based design, compared to twelve in a crossbar.

Despite its theoretical elegance, Valiant’s original EUG has not, to our knowledge, been implemented in hardware. Recent works on EUG [4], [5] still focus on generating circuits and configurations from the EUG, but do not address the

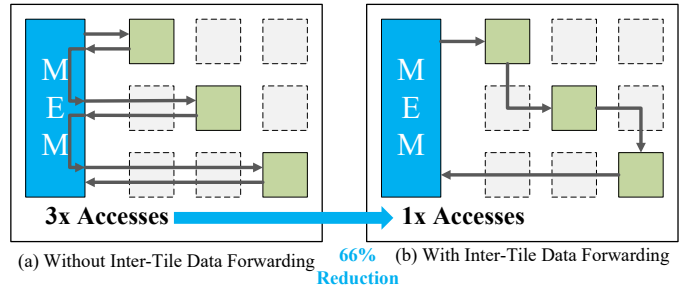


Fig. 2. (a) Accelerator without data forwarding: tiles communicate with off-chip memory independently. (b) Accelerator with data forwarding: data can be forwarded directly between tiles instead of via memory.

implementation of the circuit after synthesis or place-and-route. The key challenge lies in its purely combinational structure, which results in prohibitively long critical paths and lacks a temporal framework for timing analysis.

To address this, we introduce *Temporal-Zone Timing* (TZT), which transforms an EUG into a practical Timed EUG. We formally prove that TZT preserves functional equivalence under arbitrary task-to-tile mappings, while significantly reducing the critical path length; based on this, we present FUSE, a DNN accelerator that integrates Timed EUG to enable fragmentation-free spatial multi-tenancy. By post-layout evaluation, compared to the state-of-the-art Planaria [1] and SARA [2], FUSE improves average QoS by $1.74\times$ and $1.38\times$, improves throughput by $1.57\times$ and $1.11\times$, and lowers area overhead by $3.44\times$ and $3.00\times$. This paper makes the following contributions:

- We show that inter-tile forwarding in multi-tenant accelerators forms bounded-degree DAGs, enabling embedding into Valiant’s EUG with asymptotically optimal complexity.
- We propose a timing method (TZT) to transform EUG into Timed EUG, and prove its functional equivalence across arbitrary spatial mappings.
- We design FUSE, the first fragmentation-free spatial multi-tenant accelerator built on Timed EUG, and validate its hardware efficiency via full place-and-route using FreePDK45 [6], exhibiting $3.00\times$ area reduction.
- We perform comprehensive evaluation and show that FUSE achieves $1.38\times$ QoS improvement and $1.11\times$ throughput improvement.

II. BACKGROUND AND MOTIVATION

In this section, we first discuss contemporary multi-tenant applications and highlight the fragmented nature of their DNN tasks, motivating the need for spatial multi-tenant accelerators. We then categorize existing accelerators based on their inter-tile interconnect topologies, analyzing the trade-off between hardware overhead and forwarding flexibility. This analysis motivates our work: eliminating fragmentation during dynamic task mapping by enabling low-cost yet universal inter-tile data forwarding, systematically improving multi-tenant performance with low area overhead for spatial multi-tenant DNN accelerators.

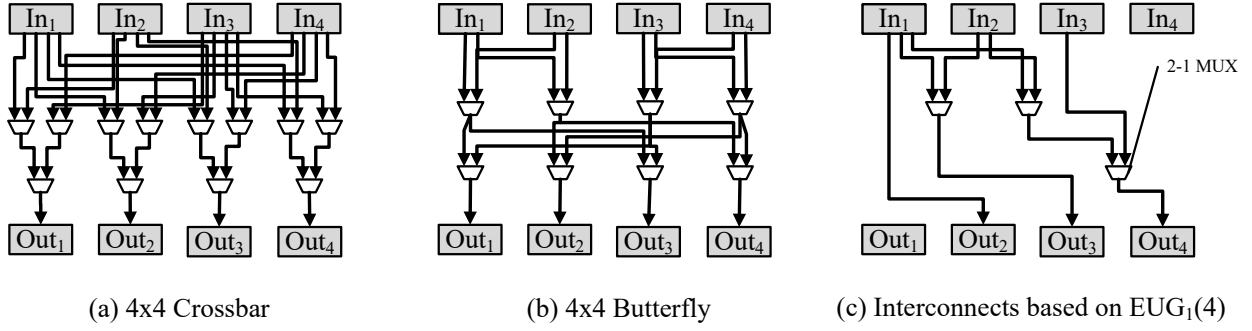


Fig. 3. Three types of universal interconnects between 4 tiles. (a) A 4×4 crossbar with 12 MUXs; (b) A 4×4 butterfly network with 8 MUXs; (c) A universal interconnect based on $EUG_1(4)$ with only 3 MUXs.

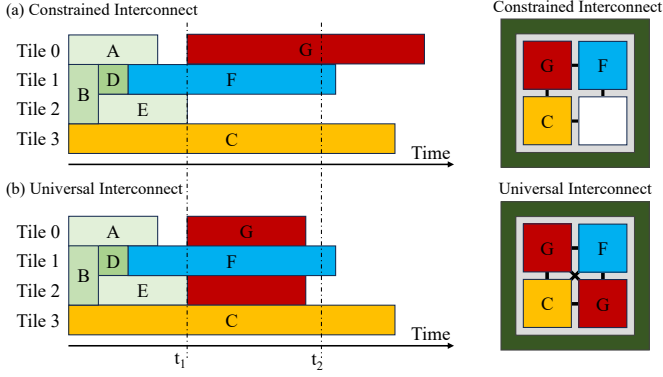


Fig. 4. Fragmentation example. (a) Universal interconnect: Task G can meet the deadline t_2 . (b) Constrained interconnect: Task G is delayed beyond t_2 due to fragmentation.

A. Multi-Tenant Applications

A broad range of multi-tenant applications has transformed daily life and work, with DNN-driven applications delivering particularly compelling user experiences. Typically deployed on portable edge devices with constrained hardware resources, these applications must execute multiple distinct DNNs concurrently under strict latency requirements. For example, in VR games, a headset—limited in both compute throughput and silicon budget—must simultaneously run gesture recognition (HandShape/Pose [7]) and eye-gaze tracking (RITNet [8]) to accurately interpret user actions. To ensure immersion, the motion-to-photon latency must remain below 15–20 ms to avoid VR vertigo [9], [10].

These workloads share two characteristics: each DNN is relatively small in size and compute demand, yet they differ significantly in intensity and structure. Thus, the challenge lies not in scaling up for massive single models but in concurrently supporting diverse, fragmented tasks under tight resource and latency constraints. This creates an urgent demand for lightweight, efficient, and highly flexible multi-tenant accelerators.

B. Multi-Tenant DNN Accelerators

Multi-tenant accelerators can be broadly categorized into temporal and spatial designs [1], [2]. Temporal accelerators, such as PREMA [11], multiplex a large systolic array ($128 \times$

128 PEs) over time to execute tasks sequentially. However, for fragmented multi-tenant workloads, temporal approaches yield poor utilization, which is often below 30% [12]–[14]. Therefore, we focus on spatial multi-tenancy, where the systolic array is partitioned into fine-grained tiles and multiple DNNs are mapped to disjoint subsets simultaneously, thereby improving utilization and scheduling flexibility.

Crucially, fine-grained spatial partitioning requires efficient inter-tile forwarding. When the tiles collaborate on a General Matrix Multiplication (GEMM) operation, activations, weights, or partial sums must be exchanged across tiles. As illustrated in Fig. 2(a), relying solely on memory-to-tile communication (i.e., without direct tile-to-tile forwarding) introduces redundant memory accesses whenever tiles need to exchange data because all inter-tile communication must traverse memory. Thus, dedicated inter-tile forwarding paths are essential to avoid redundant access, reducing 66% memory access as shown in Fig. 2(b).

Existing accelerators differ primarily in how they implement these forwarding paths to balance between hardware cost and connectivity. On one hand, designs like SOSA [15] provide universal any-to-any connectivity using complex interconnects. As shown in Fig. 3(a) and (b), traditional universal interconnects like crossbars and butterfly networks enable high flexibility but require large gate counts and wiring complexity, inflating PE array area by more than 35%. On the other hand, designs like SARA [2] (2D mesh with only nearest-neighbor transfers) and Planaria [1] (Fig. 1(a), ring buses restricted to intra-ring communication) adopt constrained topologies. While area-efficient, these limit forwarding paths, creating fragmentation when tasks cannot fully utilize idle tiles. Fig. 4 illustrates a fragmentation scenario. When Task G arrives at t_1 , Tile0 and Tile2 are idle. With a universal interconnect, Task G can exploit the direct data-forwarding path between Tile0 and Tile2 and complete before its deadline t_2 . In contrast, with a constrained interconnect, Task G cannot be jointly executed on these two tiles, leaving Tile2 idle. This resource fragmentation delays Task G beyond t_2 and degrades performance.

In summary, the key to efficiently supporting fragmented multi-tenant workloads is an inter-tile forwarding network that achieves both *high flexibility* and *low hardware overhead*, as

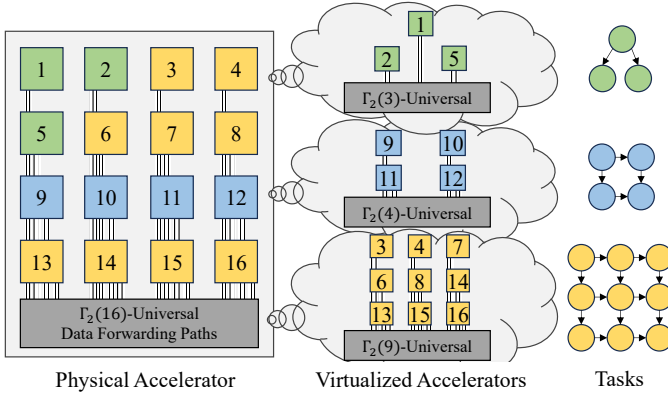


Fig. 5. DAG enables accelerator virtualization. $\Gamma_2(n)$ represents the set of all n -vertex DAGs with in-/out-degree bounded by two. The physical accelerator with $\Gamma_2(16)$ can break down to virtualized accelerators with $\Gamma_2(3)$, $\Gamma_2(4)$, and $\Gamma_2(9)$. The virtualized tiles need not be physically adjacent.

illustrated by the teaser example in Fig. 3(c). We use only 3 MUXs to implement a universal interconnect between 4 tiles, which is significantly fewer than the 8 MUXs required by the butterfly and the 12 MUXs required by the crossbar.

III. EUG-INSPIRED DATA-FORWARDING PATHS

Dynamic task mapping inevitably leaves fragmented tiles, whose positions cannot be predetermined. Eliminating this fragmentation requires non-blocking data forwarding between arbitrarily placed tiles. Conventional approaches model this requirement as *full permutation*, leading to costly interconnects such as crossbars or butterfly networks [15]. Our key insight is that **full permutation is unnecessary as multi-tenant DNN dataflows are inherently acyclic**.

A. Abstracting DNN Dataflows to DAGs

DNN computations naturally map to DAGs: vertices represent operators, and edges denote data dependencies. During compilation, each operator is assigned to an accelerator tile, and dependencies are realized through inter-tile forwarding paths. This model remains robust under standard compiler transformations—i.e., operator decomposition, task partitioning, and kernel fusion—since DAGs are closed under vertex expansion, subgraph extraction, and disjoint union. In short, any subgraph of a DAG is still a DAG, and the union of independent DAGs is also a DAG.

When inter-tile forwarding paths are designed to universally support arbitrary DAGs, the closure property of DAGs directly enables *hardware virtualization*: physical tiles can be partitioned into arbitrary subsets, with each functioning as an independent virtual accelerator. As shown in Fig. 5, a 16-tile physical accelerator can be partitioned into three virtual accelerators containing 3 (green), 4 (blue), and 9 (yellow) tiles, respectively. Each virtual accelerator runs a task whose inter-tile communication is represented by a DAG. The union of these per-virtual-accelerator DAGs is itself a DAG on the full set of 16 nodes and can be supported by the universal inter-tile forwarding paths. This property therefore allows dynamic, fragmentation-free allocation of accelerator resources.

Since DNN computations form DAGs, and compositions of DNNs preserve acyclicity, reverse forwarding paths provide no functional benefit yet impose significant hardware overhead. This observation reduces the problem to efficiently embedding arbitrary DAGs of n vertices (accelerator tiles). Valiant’s EUG [3] offers an asymptotically optimal construction for such embeddings.

B. Mapping DAGs to EUGs

Valiant’s work on embedding bounded-degree DAGs into EUGs provides the theoretical foundation for such designs. To leverage EUGs, the in-/out-degree of vertices must be bounded by intrinsic DNN dataflow properties. In practice, DNN operators consume and produce at most two data streams: matrix multiplications use activations and weights; element-wise operators use activations; systolic accumulations handle activations and partial sums. Even with broadcasts, systolic propagation ensures the out-degree remains bounded by two without loss of generality.

Thus, $\text{EUG}_2(n)$ —i.e., the EUG supporting n -vertex DAGs with in-/out-degree ≤ 2 —is sufficient to realize fragmentation-free multi-tenant DNN accelerators. This construction is asymptotically optimal in size, and significantly more compact than traditional crossbar or butterfly interconnects.

a) Core Concepts: The presentation of EUGs and our extensions introduce several definitions, symbols, and proofs. We place formal definitions inside boxed environments. Readers interested in theoretical details may refer to these boxes, while those focusing on architectural implications may safely skip them without losing conceptual understanding.

We denote by Γ the set of all DAGs, and by $\Gamma_k(n)$ the set of DAGs with n vertices, each with in-/out-degree bounded by k . In our context, dataflows in n -tile multi-tenant accelerators belong to $\Gamma_2(n)$. Because DAGs are acyclic, their vertices can always be numbered in topological order, which is essential for embedding algorithms.

Definition 1: ($\Gamma_k(n)$) Let $\Gamma_k(n)$ be the set of DAGs with n vertices and maximum in-/out-degree k . Each DAG $G = (V, E) \in \Gamma_k(n)$ assigns vertices V the labels $1, 2, \dots, n$ such that for every edge $(i, j) \in E$, $i < j$.

Implementing a DAG on an EUG is called *embedding*, where DAG vertices are mapped to EUG poles, and DAG edges are mapped to edge-disjoint paths in the EUG. No two paths traverse the same edge in the EUG.

Definition 2: (Embedding) An embedding ρ of $G = (V, E)$ into $G' = (V', E')$ is a one-to-one mapping from V to V' and from each $e = (i, j) \in E$ to a directed path $\rho(e)$ from $\rho(i)$ to $\rho(j)$ in G' , such that all paths are pairwise edge-disjoint.

An $\text{EUG}_k(n)$ is a graph that can embed any DAG from $\Gamma_k(n)$.

Definition 3: (Edge-Universal Graph) A graph G is an EUG for $\Gamma_k(n)$ if it has distinguished poles $p_1, \dots, p_n$

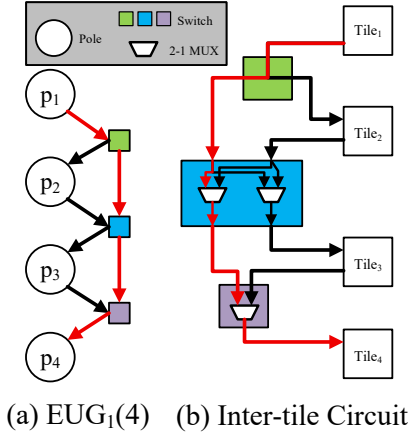


Fig. 6. (a) Graph $EUG_1(4)$. (b) The inter-tile circuit between four tiles derived from $EUG_1(4)$.

such that, for any $G' \in \Gamma_k(n)$ and any labeling, there exists an embedding ρ with $\rho(i) = p_i$. We denote such a graph as $EUG_k(n)$.

EUGs contain two types of vertices: *Poles*, where accelerator tiles are mapped, and *Switches*, which route embedded paths.

b) Construction Principle. As shown in Fig. 6, an EUG can be directly realized as an inter-tile circuit: poles become tiles, switches become multiplexers (MUXs), and edges become wires. Thus, a path in the EUG (e.g., the red path from p_1 to p_4 in $EUG_1(4)$) maps directly to a circuit path for data forwarding (the red path from $Tile_1$ to $Tile_4$ in the inter-tile circuit). Concretely, there are three types of switches in an EUG: 1-2 switches (fan-in 1, fan-out 2), shown as a green switch in (a); 2-2 switches, shown as a blue switch in (a); and 2-1 switches, shown as a purple switch in (a). We implement a 1-2 switch with two outgoing wires in the parent node, a 2-2 switch with two 2:1 MUXs, and a 2-1 switch with one 2:1 MUX. Consequently, any path in the EUG that embeds an edge e from the dataflow DAG corresponds to a circuit path in the interconnect, realizing the data forwarding specified by the dataflow DAG.

A $EUG_k(n)$ can be constructed recursively using two rules, based on the basic $EUG_1(2)$ block shown in Fig. 7(a). Unlike the traditional EUG, the $EUG_1(2)$ block includes additional vertices called *recursion nodes*, which serve as connection points for paths entering and leaving the block.

- **Rule 1:** Construct an $EUG_1(n)$ by chaining basic $EUG_1(2)$ blocks as a skeleton, and connect the recursion nodes using two $EUG_1(\lceil \frac{n}{2} \rceil - 1)$ highway.
- **Rule 2:** Construct an $EUG_k(n)$ by replicating $EUG_1(n)$ k times.

Fig. 7(b) illustrates an $EUG_2(8)$ built with these rules and Fig. 7(c) is the corresponding inter-tile circuit. An $EUG_2(8)$ is constructed by duplicating two $EUG_1(8)$ (left and right)

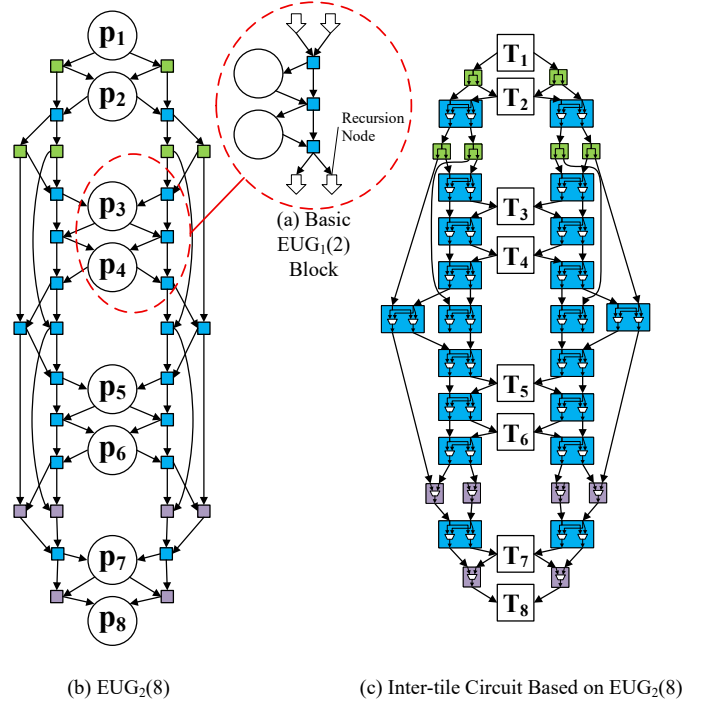


Fig. 7. (a) The basic block $EUG_1(2)$. (b) Recursive construction of $EUG_2(8)$ using $EUG_1(2)$. (c) Translated inter-tile circuit based on $EUG_2(8)$.

following Rule 2. Each $EUG_1(8)$ is itself built by chaining four $EUG_1(2)$ blocks as a skeleton and connecting the recursion nodes via two $EUG_1(3)$ highways. The resulting $EUG_2(8)$ is then translated into an inter-tile circuit to realize universal inter-tile data forwarding. Note that this recursive construction illustrates the principle rather than the minimum design. More efficient constructions with fewer switches exist [4], [5], and for small n , optimal EUGs can be manually designed or enumerated.

After constructing $EUG_k(n)$, we generate a switch configuration that embeds a target DAG $G \in \Gamma_k(n)$ by following the embedding algorithm from a recent EUG work [5]. From that switch configuration we derive the select signals for all multiplexers in the inter-tile circuit.

Tasks can be mapped onto the EUG at any time during dynamic task scheduling. Fig. 8(a) shows a timeline during dynamic scheduling of four tasks and highlights two time instants, t_1 and t_2 , with different task-to-tile allocations. Fig. 8(b) shows the task mapping at t_1 : the 8-tile accelerator is partitioned into two virtual accelerators—six (green) tiles for $Task_1$ and two (red) tiles for $Task_2$ —and two DAGs that represent the inter-tile communication within these virtual accelerators. The union of these DAGs is embedded into $EUG_2(8)$ with matching colors, so the mapping of $Task_1$ and $Task_2$ are supported by the inter-tile circuit derived from $EUG_2(8)$ (see Fig. 7(c)). Fig. 8(c) shows the mapping at t_2 in the same manner. Together, these examples demonstrate that an 8-tile accelerator with an $EUG_2(8)$ -based interconnect supports arbitrary task mappings, since $EUG_2(8)$ can embed any dataflow DAG on eight vertices that represents inter-tile

¹Note that the term “switch” here refers to a graph-level routing node and differs from the conventional notion of a circuit switch.

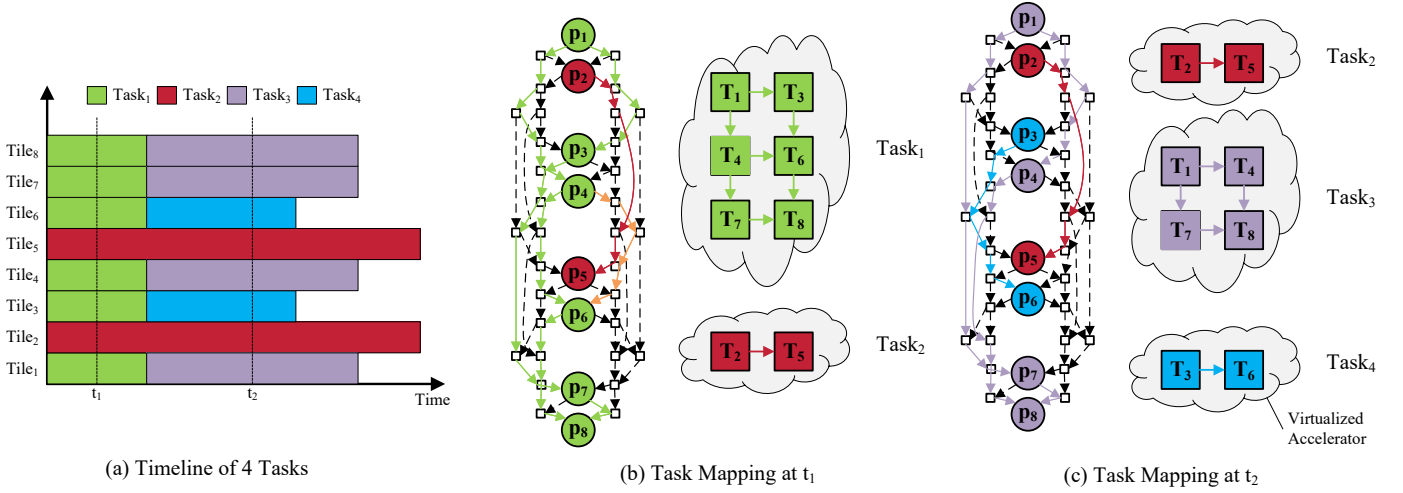


Fig. 8. Dynamic task mapping on $EUG_2(8)$. The $EUG_2(8)$ embeds the dataflow DAGs of tasks at t_1 (b) and t_2 (c) during dynamic task scheduling (a).

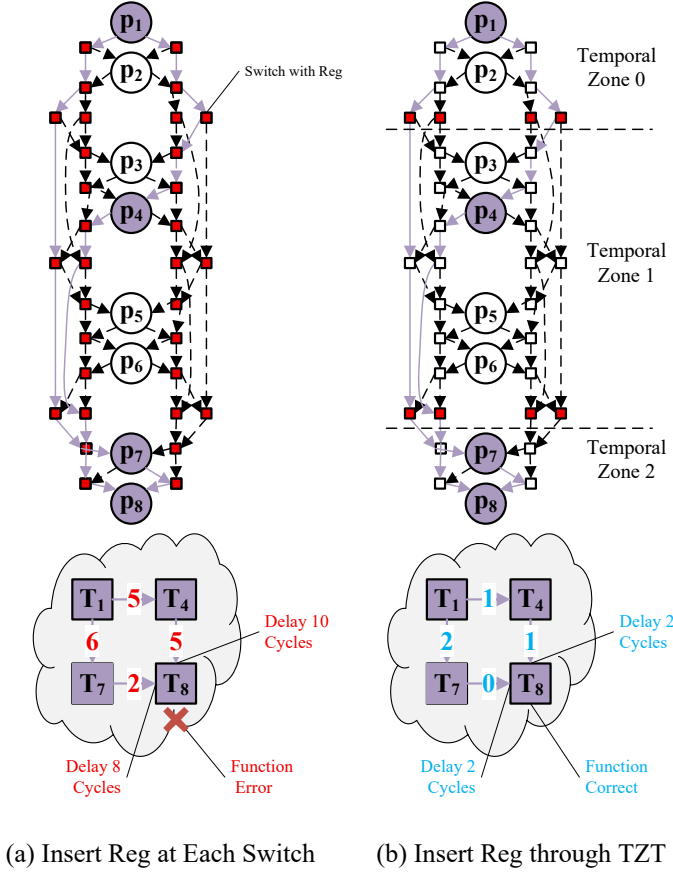


Fig. 9. Timing Methods for $EUG_2(8)$. (a) Naive register insertion adds a register after every switch. (b) Our TZZ method aligns the effective delay for all inputs to each tile.

communication.

However, direct application of EUG on the multi-tenant DNN accelerator still faces substantial challenges in timing that prior works have not addressed. We thus introduce a timing method for EUGs in Sec. IV.

IV. TIMED EUG VIA TEMPORAL-ZONE TIMING

Building on the theoretical foundation of the EUG, we now address the principal barrier to deploying it as a universal data-forwarding fabric in spatial multi-tenant DNN accelerators: *bounding the critical-path delay*. In its original formulation, the EUG is purely combinational—signals can traverse arbitrarily long paths without synchronization. This unbounded propagation time results in impractically long critical paths, which prior work has not addressed yet.

A naive way is to insert a pipeline register after every switch to shorten individual path segments. While this reduces the length of any one combinational segment, it introduces inconsistent delays between different input paths to the same destination tile. As shown in Fig. 9(a), when we insert registers after every switches in $EUG_2(8)$, two inputs to Tile₈ experience delays of 10 (pass through 10 red switches along Tile₁ → Tile₄ → Tile₈) and 8 cycles (pass through 8 red switches along Tile₁ → Tile₇ → Tile₈), respectively, causing data to arrive out of sync.

To solve this, we turn to the *Retiming Lemma* of Leiserson and Saxe [16], which provides a formal condition for inserting registers while preserving the system’s synchronization.

Theorem 1:

(Retiming Lemma) Let S be a synchronous system with communication graph G , and let lag be a function mapping each vertex v of G to an integer, with the source (host) vertex mapped to zero. If for every edge $e = (i, j)$ in G , the value $w(e) + lag(j) - lag(i) \geq 0$, then replacing the weight $w(e)$ with $\hat{w}(e) = w(e) + lag(j) - lag(i)$ yields a new system $\hat{S}$ that is functionally equivalent to S .

In our context, $EUG_k(n)$ can be modeled as a synchronous system with all initial edge weights $w(e) = 0$. The challenge is to choose a *lag function* that satisfies the lemma’s non-negativity constraint while enforcing balanced delays to each

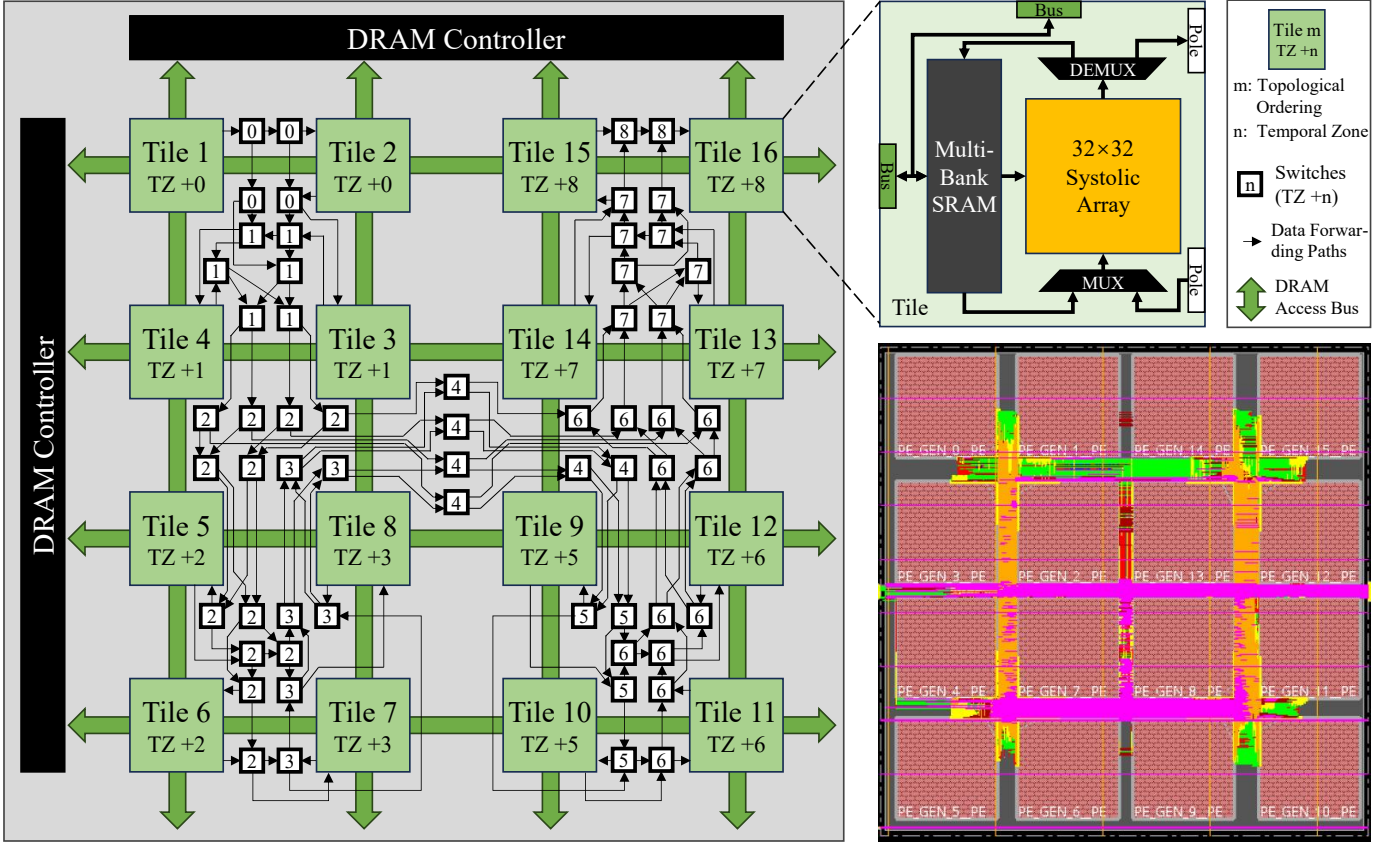


Fig. 10. FUSE architecture. FUSE integrates 16 accelerator tiles, each with local SRAM and a systolic array. Tiles connect to off-chip DRAM via 8 buses and to each other through a timed $EUG_2(16)$ interconnect.

tile. We introduce the TZT method as a structured retiming procedure tailored to EUG-based interconnects.

Lemma 1: (Temporal-Zone Timing Method) Given $EUG_k(n) = (V_n, E_n)$, define a function $lag : V_n \rightarrow \mathbb{N}$ that is monotonically non-decreasing along the topological order of V_n . For each edge $e = (v_i, v_j)$, insert exactly $lag(v_j) - lag(v_i)$ registers.

Proof of Lemma 1: Since lag is monotonic along the DAG's topological direction, $lag(v_j) \geq lag(v_i)$ for every edge, ensuring non-negative register counts. By Lemma 1, the timed $EUG_k(n)$ is functionally equivalent to the original.

Corollary 1: For any task DAG G embedded in $EUG_k(n)$ via ρ , we can set $lag_G(v_i) = lag_{EUG}(\rho(v_i))$ to guarantee that G inherits the same timing alignment, preserving functional correctness.

According to Lemma 1, we can partition $EUG_k(n)$ into multiple temporal zones, with vertices in each zone having identical lag values, i.e. $lag(v_i) = TZ(v_i)$, where $TZ(v_i)$ denotes the temporal zone index of v_i . This segments the prohibitively long critical path into multiple pipeline stages, while preserving functional correctness. For each task's dataflow graph G embedded into the EUG, partitioning is performed using the same lag function applied when partitioning the

EUG. Since the embedding from G to the EUG follows the topological order, G also preserves functional correctness.

Fig. 9 (b) illustrates an example where $EUG_2(8)$ is partitioned into three temporal zones: both data streams arriving at Tile₈ are delayed by 2 cycles, avoiding inconsistent data-forwarding delays and preserving functional correctness. The universal inter-tile network is then constructed based on the timed EUG using the TZT method.

V. AN ACCELERATOR INTEGRATING TIMED EUG

Leveraging the proposed timed EUG, we design FUSE, a fragmentation-free spatial multi-tenant DNN accelerator. In this section, we describe both the *hardware architecture* and the *software stack* that together realize FUSE.

A. FUSE Hardware Architecture

Fig. 10 illustrates the FUSE architecture. It consists of 16 accelerator tiles, each capable of executing independently or as part of a dynamically allocated virtual accelerator. Each tile integrates: (1) a 32×32 systolic array supporting input-weight/output-stationary dataflows; (2) local memory with six single-port SRAM banks, providing 32-word/cycle bandwidth to sustain MAC throughput and enable double-buffering; (3) a local control unit that executes instructions from the global scheduler and directs operand movement between memory,

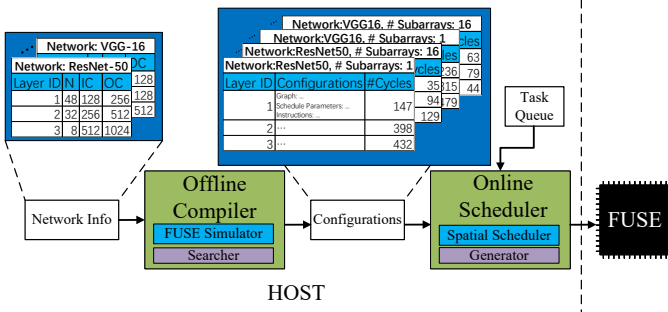


Fig. 11. FUSE software workflow. An offline compiler pre-compiles DNNs into candidate configurations. An online scheduler allocates tasks across tiles and generates instructions and switch configurations for FUSE hardware.

systolic array, and network; (4) an I/O port connected to the off-chip memory bus; (5) a routing port connected to the EUG-based inter-tile network. Tiles communicate with off-chip DRAM via eight buses connected to eight DRAM channels, as indicated by the green arrows in Fig. 10. Each bus is shared by the tiles in the same row or column.

The inter-tile network follows the EUG₂(16) construction. It is realized using 32-word-wide switches (two MUXs per switch) that can be reconfigured every cycle via 1-bit control registers. HOST can write these registers through the IO port in TZ+0. Pipeline registers are inserted according to the TZT method, bounding critical paths while preserving functional equivalence.² Fig. 10 shows the temporal zone of each tile and switch in our timing of FUSE.

a) **Network Reconfiguration:** The inter-tile network can be reconfigured to support different task mappings by writing the control registers through the IO port. This reconfiguration does not affect tasks unchanged during the reconfiguration (e.g., $Task_2$ in Fig. 8). For any data packet associated with such a task, it traverses either the pre-reconfiguration path or the post-reconfiguration path, both of which support the required data communication.³ For tasks changed during the reconfiguration, execution must wait until the entire network is fully reconfigured. This process takes 9 cycles because FUSE is partitioned into 9 temporal zones, as shown in Fig. 10.

b) **Layout optimizations:** While TZT partitions paths into pipeline stages, the area of individual tiles ($1mm^2$ under FreePDK45) results in extended physical distances between tiles within the same temporal zone. This long distance can still create intra-zone propagation delay, limiting the accelerator’s operating frequency. To mitigate this, we exploit two layout optimizations: (1) Hilbertian placement: Tiles (i.e.

²For the control register of each switch in TZ+n, we insert n pipeline registers in the path from the IO port to it.

³More concretely, suppose the tile located at TZ+n emits a packet at time t_1 , and the inter-tile network at TZ+n is reconfigured at time t_2 . Then, for any $m \geq n$, the packet arrives at TZ+m at time $t_1 + (m - n)$, while the inter-tile network at TZ+m is reconfigured at time $t_2 + (m - n)$. If $t_1 \geq t_2$ (i.e. the packet is emitted after reconfiguration), and thus

$$t_1 + (m - n) \geq t_2 + (m - n)$$

for any m , meaning that the packet follows the post-reconfiguration path. Otherwise, it follows the pre-reconfiguration path.

poles in EUG) constitute a topologically ordered 1D chain that is partitioned into temporal zones. To optimize intra-zone propagation delays, tiles are arranged along Hilbert-curve trajectories, leveraging the spatial locality of Hilbert-curve to reduce physical distances between tiles within the same zone. (2) Tile rotation: We rotate tiles to reduce physical distances between routing port of tiles within the same zone, further optimizing intra-zone propagation delays.

B. FUSE Software Stack

The FUSE software stack consists of an offline compiler and an online scheduler, as shown in Fig. 11. To execute workloads on FUSE, the offline compiler first precompiles all DNN models, and the online scheduler then dynamically assigns tasks to the FUSE accelerator to enable efficient multi-task execution. Both the offline compiler and the online scheduler run on the host system.

a) **Offline Compiler:** Leveraging the fact that the universal inter-tile interconnect in FUSE can embed any possible dataflow DAG, the offline compiler processes all potentially executable DNN models and searches for the optimal dataflow DAG for executing each model on a virtualized accelerator with any possible number of tiles (from 1 to 16) using the FUSE simulator. Specifically, the compiler performs an exhaustive search over the candidate dataflow DAGs and evaluates their performance with the simulator to identify the optimal one. In addition to the *dataflow DAG*, which describes the data-forwarding patterns across tiles, the compiler also generates: *schedule parameters*, including loop order, tiling factors, and other scheduling decisions, which are also obtained through the exhaustive search; and a set of *instructions* for the local control units in each tile, including *load*, *store*, *mmul*, and other instructions required to execute the scheduled computation.

b) **Online Scheduler:** At runtime, the online scheduler dynamically allocates tasks to tiles and integrates the offline-compiled configurations with live system information. Specifically, a spatial scheduling policy (which can be customized by the user) determines the virtual accelerator for each task in the task queue. As tasks in the queue typically have unsynchronized execution progress, the scheduler makes allocation decisions on a per-task basis rather than batching multiple tasks together. In our experiments, FUSE adopts a QoS-aware spatial scheduling policy similar to Planaria [1]. The scheduler estimates the minimum tile allocation required for each queued task to meet its QoS requirement; when all tasks fit, it assigns these allocations and distributes spare tiles using a priority- and remaining-time-based score. When resources are insufficient, it ranks tasks using a score based on priority, predicted slack, and minimum tile demand, assigning tiles to higher-scored tasks first and deferring lower-scored tasks. Since this scheduling process is performed only when a task arrives or when a running task is estimated to finish, and its average runtime is only 0.13 μs as measured on an Intel Xeon Gold 6230 CPU @ 2.10GHz, we overlap it with task execution and treat its runtime overhead as negligible.

TABLE I
OUR EXPERIMENTAL WORKLOADS BASED ON XRBENCH [17].

Workload	Task	Model	FPS
AR Assistant	Speech Recognition	Emformer [18]	3
	Depth Estimation	Midas [19]	30
	Keyword Detection	Key-Res-15 [20]	3
	Object Detection	D2Go [21]	10
	Plane Detection	PlaneRCNN [22]	15
	Semantic Segmentation	HRViT [23]	10
AR Gaming	Depth Estimation	Midas [19]	30
	Hand Tracking	Hand Shape/Pose [7]	45
	Plane Detection	PlaneRCNN [22]	15
VR Gaming	Eye Segmentation	RITNet [8]	60
	Hand Tracking	Hand Shape/Pose [7]	45
	Gaze Estimation	Eyecod [24]	60
Outdoor Activity A	Speech Recognition	Emformer [18]	3
	Keyword Detection	Key-Res-15 [20]	3
	Object Detection	D2Go [21]	30
	Semantic Segmentation	HRViT [23]	10
Outdoor Activity B	Speech Recognition	Emformer [18]	3
	Keyword Detection	Key-Res-15 [20]	3
	Object Detection	D2Go [21]	30
Social Interaction A	Depth Refinement	Sparse-to-Dense [25]	30
	Eye Segmentation	RITNet [8]	60
	Gaze Estimation	Eyecod [24]	60
	Hand Tracking	Hand Shape/Pose [7]	30
Social Interaction B	Action Segmentation	TCN [26]	30
	Eye Segmentation	RITNet [8]	60
	Gaze Estimation	Eyecod [24]	60

Once tile allocation is determined, a configuration generator: (1) consults the configuration table produced by the offline compiler; (2) generates the complete dataflow DAG that describes the data-forwarding patterns across all tiles in FUSE for the current scheduling state; and (3) produces the instructions for each tile. The configuration of the inter-tile EUG₂(16) network is then generated by embedding this DAG into the EUG. Similar to the scheduling process, we also overlap EUG embedding with task execution. Since it takes only 20.24 us on average as measured on an Intel Xeon Gold 6230 CPU @ 2.10GHz, we treat its runtime overhead as negligible.

VI. EXPERIMENTAL METHODOLOGY

This section details the FUSE implementation, the workloads and metrics for evaluation, and the baseline multi-tenant accelerators against which we compare FUSE’s performance.

A. Workloads

In our evaluation, we select AR/VR workloads derived from the XRBench benchmark suite [17], as summarized in Table I. XRBench provides seven Real-Time Multi-task Multi-model (RTMM) workloads based on real-world industrial use cases from production scenarios. To simulate multi-tenant execution, we configure each computational task to follow a periodic arrival pattern according to its Frames Per Second (FPS) as specified by XRBench. Furthermore, we set the latency target of each task to the reciprocal of its FPS (i.e., $\frac{1}{\text{FPS}}$, measured in seconds). To evaluate FUSE’s performance under different

TABLE II
FUSE HARDWARE CONFIGURATION IN THE EVALUATION.

Parameter	Value
Systolic array dimension (per tile)	32×32
SRAM size (per tile)	768KB
Number of tiles	16
Total on-chip SRAM size	12MB
DRAM bandwidth	64GB/s
Frequency	400MHz

workload intensities, we use three levels of QoS: (i) QoS-L, a light target ($1 \times$ the FPS in XRBench); (ii) QoS-M, a medium target ($2 \times$ the FPS), representing more intensive tasks and stricter latency constraints; and (iii) QoS-H, a hard target ($3 \times$ the FPS), representing the most intensive tasks and the strictest latency constraints. Compiling all models in the XRBench takes 22307 seconds in our experiment.

B. FUSE Implementation

We implement the FUSE hardware in Verilog HDL and evaluate its area and latency after place-and-route using the FreePDK-45nm standard cell library. Fig. 10 shows the post place-and-route layout of FUSE. On-chip SRAM area and power metrics are obtained using Cacti-P [27]. Table II summarizes the hardware configuration used in our evaluation of FUSE. For a fair comparison, we use the same scheduling algorithm as Planaria [1] as the custom scheduling algorithm in the FUSE online scheduler.

We also develop a cycle-accurate FUSE simulator for fast offline compilation. We first extract timing characteristics from Innovus under FreePDK-45nm [6], and simulate the RTL of the tiles and interconnects on compiled typical traces to generate VCD files. We then use the execution cycles observed in the VCD files to calibrate the cycle counts of different FUSE modules, such as the systolic array, under various tasks. The memory subsystem is modeled using CACTI-P [27]. We further model a 9-cycle overhead for network reconfiguration, according to Sec. V-A. Finally, we construct an event-driven framework to orchestrate these modules and accelerate simulation. We find that including the reconfiguration overhead does not lead to a measurable throughput change in any workload set.

C. Metrics

To quantitatively evaluate the multi-tenant performance of FUSE, we use the following metrics:

- **SLA Satisfaction Rate:** defined as the fraction of tasks in the workload that meet their latency targets specified in the Service-Level Agreements (SLA), i.e., the QoS latency target. It is formally expressed as:

$$\text{SLA satisfaction rate} = \frac{|\{t_i \in \mathcal{W} \mid \tau_i^{\text{actual}} \leq \tau_i^{\text{target}}\}|}{|\mathcal{W}|} \quad (1)$$

where $\mathcal{W}$ denotes the set of tasks in the workload, τ_i^{actual} represents the observed latency, and τ_i^{target} specifies the QoS latency target for task t_i .

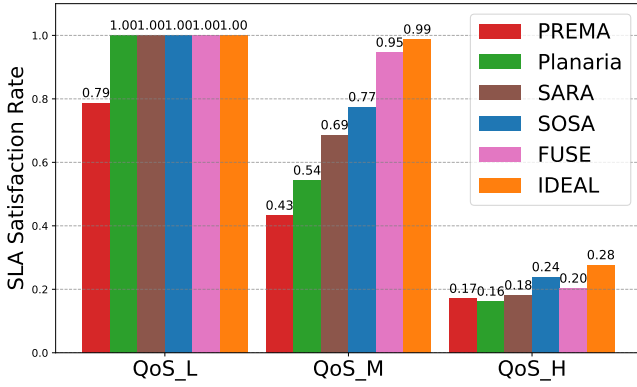


Fig. 12. Improvement in SLA satisfaction rate achieved by FUSE over multi-tenant DNN accelerator baselines under 3 QoS levels.

- **Throughput:** defined as the maximum times of FPS that accelerator achieve 95% SLA satisfaction rate.
- **Energy Efficiency:** defined as the throughput divided by the power consumed when the accelerator operates at that throughput.

D. Baselines

We evaluate FUSE against the following multi-tenant DNN accelerator baselines:

- **PREMA [11]:** a temporal multi-tenant DNN accelerator, which supports multi-tenant execution through temporal partition of compute resources.
- **Planaria [1]:** it employs a ring-based inter-tile network, using two rings to connect tiles within the accelerator. Planaria only provides non-blocking inter-tile data forwarding paths between tiles that are directly connected by the rings.
- **SARA [2]:** it employs a 2D mesh-based inter-tile network, which only provides non-blocking inter-tile data forwarding paths between adjacent tiles.
- **SOSA [15]:** a spatial multi-tenant DNN accelerator, utilizing a butterfly interconnection topology to multi-cast operands across multiple tiles.

All baselines are evaluated using the same hardware configuration as FUSE, as detailed in Table II. In addition, we implement an accelerator baseline with an **Ideal Interconnect** to provide an idealized reference point. This baseline assumes that on-chip tiles are connected by an area-free network that enables zero-latency data transfer between any pair of tiles, with no bandwidth limitation or blocking. This baseline is not intended to represent a practical implementation, but rather to quantify the performance under idealized inter-tile communication.

VII. EVALUATION

In this section, we evaluate FUSE against four baselines in Sec. VI-D to demonstrate that FUSE achieves superior performance and energy efficiency with reduced area overhead. We evaluate FUSE’s multi-tenant execution performance using seven sets of workloads in Table I, each with three QoS targets (QoS-H/M/L: hard / medium / light latency target).

A. SLA satisfaction rate Analysis

Fig. 12 shows that FUSE achieves higher SLA satisfaction rates than all four multi-tenant accelerator baselines. Under QoS-L most accelerators reach near-100% SLA satisfaction, except PREMA which attains 79%. Under QoS-H no accelerator achieves an SLA satisfaction rate above 30%. The largest differences appear at QoS-M: in the geometric-mean across the seven workloads, FUSE achieves $1.38\times$ the SLA satisfaction rate of SARA, $1.74\times$ that of Planaria, $2.19\times$ that of PREMA, and $1.18\times$ that of SOSA. Compared with the Ideal Interconnect baseline, FUSE achieves 100%, 95.83%, and 73.35% of its SLA satisfaction rate under QoS-L, QoS-M, and QoS-H, respectively.

The slightly lower SLA satisfaction rate of FUSE than SOSA under QoS-H is mainly due to the stricter timing constraints in this setting. FUSE’s scheduling policy relies on predicted execution times to determine whether tasks can meet their target deadlines. However, since inter-task DRAM channel contention cannot be accurately predicted at compile time, the predicted execution time of some tasks can be smaller than their actual runtime. As a result, these tasks may fail to complete before their target deadlines even though they were expected to do so by the scheduler. This effect is amplified under QoS-H, where the deadline constraints are much tighter, leading to a noticeable degradation in SLA satisfaction.

B. Throughput

SLA satisfaction rate is highly correlated with the chosen QoS level, which undermines its reliability as a standalone comparison metric. Thus, we define throughput as the maximum times of the FPS (i.e., QoS level) at which an accelerator achieves a 95% SLA satisfaction rate. This throughput metric more robustly reflects multi-tenant performance across varying workload intensities. Fig. 13 shows normalized throughput for all workloads (and the geometric mean) plotted against area overhead for the evaluated spatial multi-tenant accelerators, with throughputs normalized to FUSE.

PREMA consistently exhibits the lowest throughput with zero area overhead: as a temporal multi-tenant accelerator using a single large systolic array (128×128), it suffers from low utilization and thus low throughput. SOSA achieves the highest throughput among the baselines because it uses a butterfly interconnect, but this comes with the largest area overhead. Note that SOSA is not the best baseline for the AR Assistant workload: SOSA’s multicast mechanism does not effectively mitigate partial-sum accumulation traffic, whereas the offline compiler found that partial-sum forwarding (as supported by FUSE) is critical for DNNs in that workload. SARA and Planaria use constrained interconnect topologies that reduce area overhead but lead to throughput degradation. The Pareto frontier of the baselines is marked in the figure to highlight the trade-off between normalized throughput and area overhead.

In contrast, FUSE achieves throughput comparable to SOSA while using less area than SOSA, SARA and Planaria. Although both FUSE and SOSA provide universal connectivity,

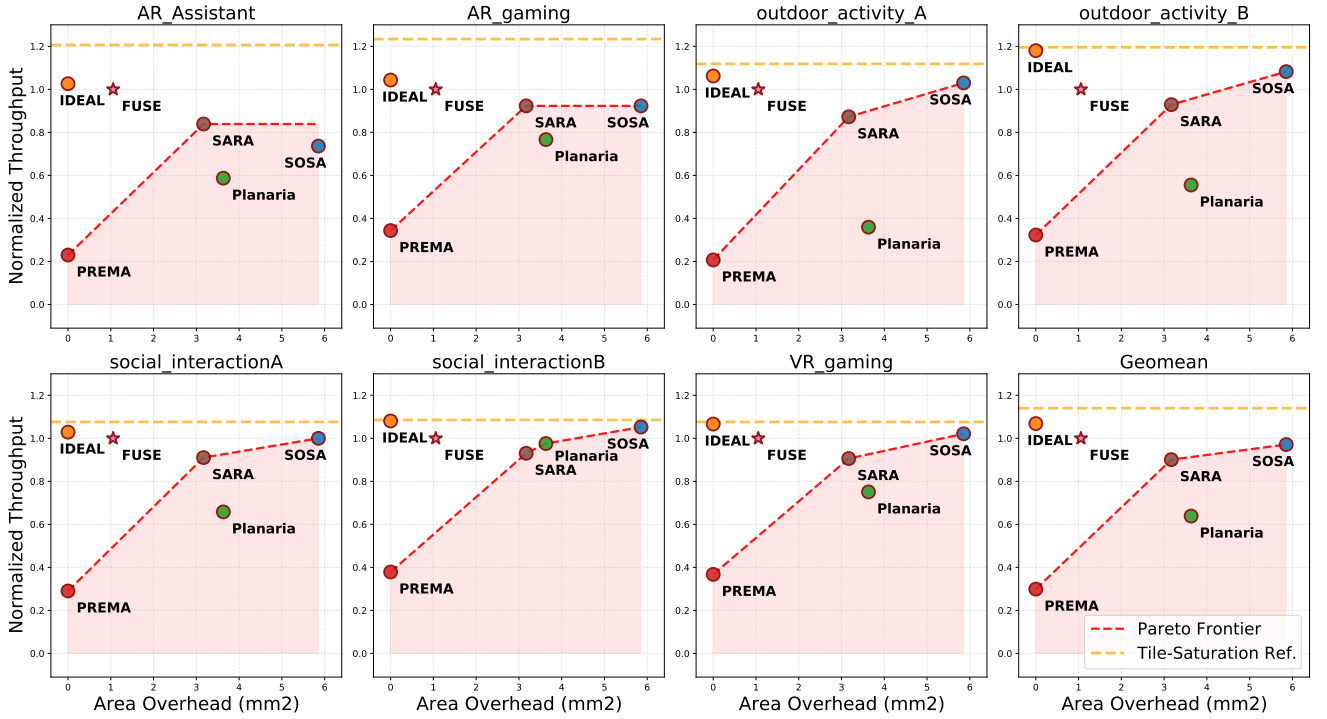


Fig. 13. Throughput and area overhead of multi-tenant accelerators. Compared to prior works, FUSE achieve higher throughput with lower area overhead.

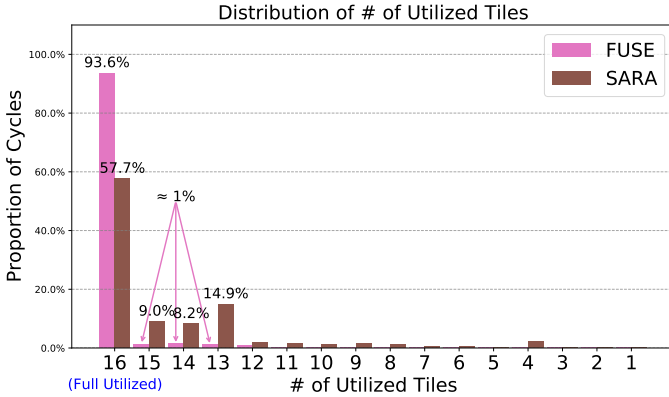


Fig. 14. Distribution of tile utilization across effective cycles for FUSE and SARA. FUSE achieves full utilization (all tiles active) in 93.6% of effective cycles.

their performance differs because SOSA uses multi-cast to reduce redundant memory access, whereas FUSE’s EUG-based design implements explicit tile-to-tile data forwarding. As a result, FUSE and SOSA exhibit different workload sensitivities. In geometric-mean results, FUSE achieves a throughput-to-area-overhead ratio $3.33\times$ that of SARA, $5.39\times$ that of Planaria, and $5.71\times$ that of SOSA. As for raw throughput, FUSE attains $1.11\times$, $1.57\times$, $3.34\times$, and $1.03\times$ compared to SARA, Planaria, PREMA, and SOSA, respectively.

We also include two reference results. The **Ideal Interconnect** baseline removes interconnect constraints by assuming zero-latency, unlimited-bandwidth, and non-blocking inter-tile communication. It serves as an idealized upper bound for quantifying the throughput gap due to practical interconnect

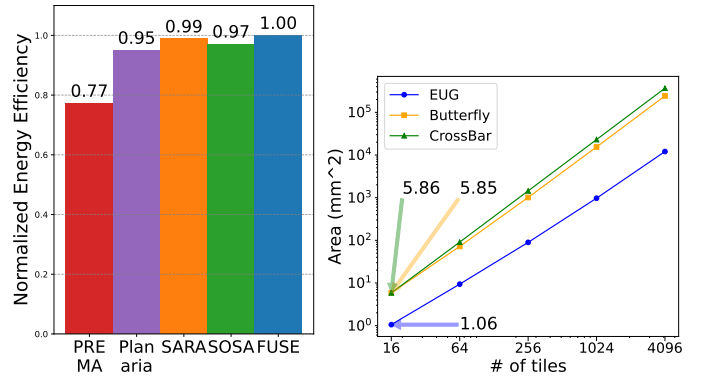


Fig. 15. Normalized energy efficiency of different multi-tenant works with different topologies for different numbers of tiles.

limitations. The **Tile-Saturation Reference** further estimates the throughput that would be achieved if the Ideal Interconnect baseline could fully utilize all tiles. Compared with the Ideal Interconnect baseline and the Tile-Saturation Reference, FUSE achieves 93.58% and 87.71% of their geometric-mean throughput, respectively.

C. Energy Efficiency

Fig. 15 shows that FUSE achieves the highest energy efficiency among all multi-tenant accelerators. Planaria and SARA achieve slightly lower energy efficiency than FUSE because of their lower throughput, while their static power consumption remains unchanged. SOSA has lower energy efficiency because its complex inter-tile network consumes a significant amount of energy. PREMA achieves the lowest

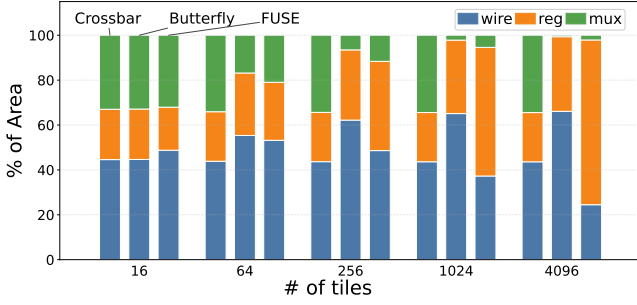


Fig. 17. Area breakdown of universal inter-tile networks with different topologies and numbers of tiles.

energy efficiency because its throughput is much lower than that of the other accelerators.

D. Utilization

To demonstrate full-tile utilization, we measure how many tiles are actively used (from 1 to 16) for each cycle across all three QoS levels and seven workloads. We then compute the proportion of cycles at each utilization level relative to the effective operating cycles, where effective operating cycles are defined as cycles in which at least one task has arrived but not yet completed (i.e., number of utilized tiles > 0).

Fig. 14 shows the distribution of utilization rates for FUSE and SARA. FUSE achieves full utilization (all tiles active) in 93.6% of effective cycles; the remaining cycles correspond primarily to brief gaps during task switching. In contrast, SARA attains full utilization in only 57.2% of effective cycles, reflecting underutilization caused by fragmentation due to its constrained 2D-MESH interconnect.

E. Critical Path Delay Analysis

We analyze the critical path delay of our FUSE hardware after place-and-route with FreePDK45. The timing report shows that the critical path delay in our universal inter-tile network, based on the timed EUG₂(16), is 2 ns, while the critical path delay within a tile is 2.5 ns. It demonstrates that the data forwarding network is no longer the frequency bottleneck, highlighting the advantages of our timing method.

F. Area Breakdown & Overhead Analysis

We analyze the area after place-and-route of our FUSE hardware using FreePDK45. We also evaluate the area of SARA, Planaria, and SOSA, the latter of which achieves universal inter-tile connectivity based on a butterfly topology. Our results show that only 6.44% (1.06 mm²) extra area in FUSE is used for the universal inter-tile network, compared to 19.33% (3.17 mm²) in SARA, 22.15% (3.63 mm²) in Planaria and 35.72% (5.85 mm²) in SOSA, achieving a 3.00 $\times$, 3.44 $\times$ and 5.55 $\times$ reduction in area overhead, respectively. We also estimate the area overhead for scaling to 7 nm based on existing work [28] [29], which are 4.39% (0.042 mm²) in FUSE, 7.75% (0.075 mm²) in SARA, 9.14% (0.088 mm²) in Planaria and 25.3% (0.24 mm²) in SOSA, respectively.

We further break down the FUSE accelerator area into tile area, DRAM-channel area, and inter-tile interconnect area.

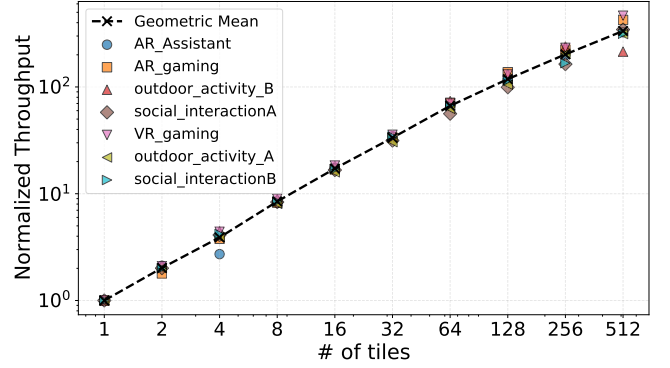


Fig. 18. Throughput of FUSE across different numbers of tiles.

The 16 tiles occupy 16.39 mm² (76.04%), the eight DRAM channels occupy 4.11 mm² (19.07%), and the FUSE inter-tile interconnect occupies 1.06 mm² (4.89%). If each tile were reduced from a 32 $\times$ 32 systolic array to a single PE while keeping the total number of on-chip PEs unchanged, the number of tiles would increase by 32 $\times$ 32=1024 $\times$. In this case, the total tile area and DRAM-channel area would remain approximately unchanged, but the FUSE interconnect area would increase from 1.06 mm² for 16 coarse-grained tiles to 32.10 mm².

G. Scalability

To analyze the scalability of our universal inter-tile network based on the EUG, we evaluate its area after synthesis and place-and-route (PNR). The reported area at 16 tiles is derived directly from PNR results. For comparison, we also evaluate the area of two other universal networks: Butterfly [15] and Crossbar. As shown in Fig. 16, our network (1.06 mm²) achieves a 5.52 $\times$ and 5.53 $\times$ reduction in area compared to the Butterfly (5.85 mm²) and Crossbar (5.86 mm²) topologies.

To project area scaling to larger tile counts, we decompose the total network area into three components: multiplexers, registers and wires. Multiplexer and register areas are estimated from their synthesized per-unit areas and the corresponding component counts for each tile configuration. Wire area is estimated by deriving the half-perimeter wire length (HPWL) from the floorplan and scaling the 16-tile PNR wire area in proportion to the predicted HPWL.

Fig. 16 illustrates the area growth as the number of tiles increases (16, 64, 256, 1024, 4096), plotted on a logarithmic scale. The results show that FUSE consistently achieves smaller area than Butterfly and Crossbar because it only needs to provide data forwarding paths from tile i to tile j (for $i < j$) in the EUG, whereas the Butterfly and Crossbar topology also supports unnecessary paths from tile j to tile i , resulting in redundant area overhead.

Fig. 17 shows the area composition of multiplexers, registers, and wires for the three topologies across all evaluated tile counts. In FUSE, registers occupy an increasing area fraction as the tile count grows, since registers scale as $O(n^2)$ while multiplexers and wires scale only as $O(n \log n)$. In Butterfly, the fraction of area occupied by multiplexers decreases with

increasing tile count, since multiplexers scale as $O(n \log n)$ while both wires and registers scale as $O(n^2)$. In contrast, the relative area breakdown of Crossbar remains nearly constant across different tile counts, because multiplexers, registers, and wires all scale as $O(n^2)$.

We also evaluate the EUG embedding and power overhead of the FUSE interconnect as the number of tiles scales from 16 to 4096. The EUG embedding overhead scales from 20.24 us at 16 tiles to 83.26, 540.40, 3516.55 and 20815.48 us at 64, 256, 1024, and 4096 tiles, respectively. The interconnect power increases from 1.91 W at 16 tiles to 18.70, 175.12, 1694.12, and 17804.25 W at 64, 256, 1024, and 4096 tiles.

Fig. 18 shows the throughput of FUSE as the number of tiles increases. We evaluate FUSE across all workloads in XRBench with the number of tiles ranging from 1 to 512. The results show that FUSE throughput increases linearly with the number of tiles.

VIII. RELATED WORK

a) Temporal Multi-Tenant DNN Accelerators: Accelerators such as PREMA [11] and V10 [30] employ time-multiplexed DNN execution on a single large systolic array (e.g., 128×128). However, these architectures suffer from limited scheduling flexibility and low utilization, failing to exploit the parallelism in multi-task workloads and the accelerator's peak TOPS capacity. As a result, they deliver suboptimal performance.

b) Spatial Multi-Tenant DNN Accelerators: A broad range of DNN accelerators and their software stacks have been proposed to improve the efficiency and flexibility of neural-network execution [31]–[40]. Architectures such as Planaria [1] partition the accelerator into multiple tiles and dynamically allocate tasks to them. Planaria [1] and SARA [2] use ring and 2D-MESH networks, respectively, to enable data forwarding between tiles, eliminating redundant memory accesses. SOSA [15] employs a butterfly network to achieve universal inter-tile data forwarding. MoCA [41] uses a global memory in place of a network to achieve data forwarding across tiles and can dynamically allocate memory resources to different tasks.

c) Multi-Tenant Scheduling: Schedulers have been proposed to enhance multi-tenant performance across various platforms. DREAM [42] and Aurora [43] propose scheduling strategies for multi-accelerator systems, while SCAR [44] addresses multi-chiplet systems and Veltair [45] develops schedulers for CPUs.

d) Edge-Universal Graph: The EUG was originally proposed by Valiant in 1976 [3] to realize universal circuits. In recent years, researchers have leveraged these universal circuits to enable Private Function Evaluation (PFE) for privacy-preserving applications such as creditworthiness checking [46] and blinded policy evaluation protocols [47], [48]. Recent work has focused on reducing the number of switches in the EUG from $5n \log n$ to $4.5n \log n$ [4], [5].

IX. CONCLUSION

In this paper, we propose FUSE, a fragmentation-free spatial multi-tenant DNN accelerator based on the EUG. FUSE enables arbitrary data forwarding between tiles through a universal inter-tile network constructed from the EUG, thereby eliminating fragmentation caused by the lack of data forwarding paths in previous works. We formally prove the dynamic reconfigurability of the EUG, and apply a T2T timing method to facilitate the implementation of a high-frequency, EUG-based network. Our evaluation on XRBench workloads demonstrates that, compared with the state-of-the-art spatial multi-tenant DNN accelerator Planaria [1] and SARA [2], FUSE increases SLA satisfaction by $1.74\times$ and $1.38\times$, and throughput by $1.57\times$ and $1.11\times$, and achieves an area overhead reduction of $3.44\times$ and $3.00\times$, respectively.

REFERENCES

- [1] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 681–697.
- [2] A. Samajdar, E. Qin, M. Pellauer, and T. Krishna, "Self adaptive reconfigurable arrays (sara) learning flexible gemm accelerator configuration and mapping-space using ml," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 583–588.
- [3] L. G. Valiant, "Universal circuits (preliminary report)," in *Proceedings of the eighth annual ACM symposium on Theory of computing*, 1976, pp. 196–203.
- [4] A. Kiss and T. Schneider, "Valiant's universal circuit is practical," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 699–728.
- [5] M. Y. Alhassan, D. Günther, A. Kiss, and T. Schneider, "Efficient and scalable universal circuits," *Journal of Cryptology*, vol. 33, no. 3, pp. 1216–1271, 2020.
- [6] Freepdk45. [Online]. Available: <https://eda.ncsu.edu/freepdk/freepdk45/>
- [7] L. Ge, Z. Ren, Y. Li, Z. Xue, Y. Wang, J. Cai, and J. Yuan, "3d hand shape and pose estimation from a single rgb image," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 833–10 842.
- [8] A. K. Chaudhary, R. Kothari, M. Acharya, S. Dangi, N. Nair, R. Bailey, C. Kanan, G. Diaz, and J. B. Pelz, "Ritnet: Real-time semantic segmentation of the eye for gaze tracking," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019, pp. 3698–3702.
- [9] K. Brunnström, E. Dima, T. Qureshi, M. Johanson, M. Andersson, and M. Sjöström, "Latency impact on quality of experience in a virtual reality simulator for remote control of machines," *Signal Processing: Image Communication*, vol. 89, p. 116005, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0923596520301648>
- [10] Y. Feng, T. Ma, Y. Zhu, and X. Zhang, "Blisscam: Boosting eye tracking efficiency with learned in-sensor sparse sampling," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1262–1277.
- [11] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.
- [12] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th annual international symposium on computer architecture*, 2023, pp. 1–14.
- [13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-I. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt,

- J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datcenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12. [Online]. Available: <https://doi.org/10.1145/3079856.3080246>
- [14] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.
- [15] A. C. Yüzügüler, C. Sönmez, M. Drumond, Y. Oh, B. Falsafi, and P. Frossard, "Scale-out systolic arrays," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 2, pp. 1–25, 2023.
- [16] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," in *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*. IEEE, 1981, pp. 23–36.
- [17] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda *et al.*, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 1–20, 2023.
- [18] Y. Shi, Y. Wang, C. Wu, C.-F. Yeh, J. Chan, F. Zhang, D. Le, and M. Seltzer, "Emformer: Efficient memory transformer based acoustic model for low latency streaming speech recognition," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6783–6787.
- [19] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 3, pp. 1623–1637, 2020.
- [20] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5484–5488.
- [21] Meta. (2022) D2go. [Online]. Available: <https://github.com/facebookresearch/d2go>
- [22] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "PlanerCNN: 3d plane detection and reconstruction from a single image," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4450–4459.
- [23] J. Gu, H. Kwon, D. Wang, W. Ye, M. Li, Y.-H. Chen, L. Lai, V. Chandra, and D. Z. Pan, "Multi-scale high-resolution vision transformer for semantic segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 12 094–12 103.
- [24] H. You, C. Wan, Y. Zhao, Z. Yu, Y. Fu, J. Yuan, S. Wu, S. Zhang, Y. Zhang, C. Li *et al.*, "Eyecod: eye tracking system acceleration via flatcam-based algorithm & accelerator co-design," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 610–622.
- [25] F. Ma and S. Karaman, "Sparse-to-dense: Depth prediction from sparse depth samples and a single image," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4796–4803.
- [26] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.
- [27] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cactip: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 694–701.
- [28] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167926017300755>
- [29] A. Ceyhan, M. Jung, S. Panth, S. K. Lim, and A. Naeemi, "Impact of size effects in local interconnects for future technology nodes: A study based on full-chip layouts," in *IEEE International Interconnect Technology Conference*, 2014, pp. 345–348.
- [30] Y. Xue, Y. Liu, L. Nai, and J. Huang, "V10: Hardware-assisted npu multi-tenancy for improved resource utilization and fairness," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3579371.3589059>
- [31] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 269–284.
- [32] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning super-computer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.
- [33] Z. Du, R. Fasthuber, T. Chen, P. lenne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Computer Architecture*, 2015, pp. 92–104.
- [34] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Computer Architecture*, 2016.
- [35] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.
- [36] W. Lou, C. Wang, L. Gong, and X. Zhou, "RV-CNN: Flexible and efficient instruction set for CNNs based on RISC-V processors," in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 11719. Springer, 2019, pp. 3–14.
- [37] J. Song, Y. Zhuang, X. Chen, T. Zhi, and S. Liu, "Compiling optimization for neural network accelerators," in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 11719. Springer, 2019, pp. 15–26.
- [38] W. Du, L. Wu, X. Chen, Y. Zhuang, and T. Zhi, "Zhuque: A neural network programming model based on labeled data layout," in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 11719. Springer, 2019, pp. 27–39.
- [39] X. Chen, S. Peng, L. Jin, Y. Zhuang, J. Song, W. Du, S. Liu, and T. Zhi, "Partition and scheduling algorithms for neural network accelerators," in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 11719. Springer, 2019, pp. 55–67.
- [40] J. Xie and Y. Liang, "SPART: Optimizing CNNs by utilizing both sparsity of weights and feature maps," in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 11719. Springer, 2019, pp. 71–85.
- [41] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 828–841.
- [42] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra, "Dream: A dynamic scheduler for dynamic real-time multi-model ml workloads," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2023, pp. 73–86.
- [43] S. Kim, J. Zhao, K. Asanović, B. Nikolić, and Y. S. Shao, "Aurora: Virtualized accelerator orchestration for multi-tenant workloads," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 62–76.
- [44] M. Odema, L. Chen, H. Kwon, and M. A. Al Faruque, "Scar: Scheduling multi-model ai workloads on heterogeneous multi-chiplet module accelerators," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 565–579.
- [45] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "Veltair: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 388–401. [Online]. Available: <https://doi.org/10.1145/3503222.3507752>
- [46] K. Friksen, M. Atallah, and C. Zhang, "Privacy-preserving credit checking," in *Proceedings of the 6th ACM Conference on Electronic*

Commerce, ser. EC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 147–154. [Online]. Available: <https://doi.org/10.1145/1064009.1064025>

- [47] K. Frikken, M. Atallah, and J. Li, "Attribute-based access control with hidden policies and hidden credentials," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1259–1270, 2006.
- [48] K. B. Frikken, J. Li, and M. J. Atallah, "Trust negotiation with hidden credentials, hidden policies, and policy cycles," in *Network and Distributed System Security Symposium*, 2006. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16974710>