FLIP2M: Flexible Intra-layer Parallelism and Inter-layer Pipelining for Multi-model AR/VR Workloads

GABRIELE TOMBESI, Computer Science, Columbia University, New York, United States
JE YANG, Computer Science, Columbia University, New York, United States
JOSEPH ZUCKERMAN, Columbia University, New York, United States
DAVIDE GIRI, Computer Science, Columbia University, New York, United States
WILLIAM BAISI, Columbia University, New York, United States
LUCA CARLONI, Computer Science, Columbia University, New York, United States

Tiled accelerator architectures provide opportunities to optimize the performance of multi-model augmented and virtual reality (AR/VR) applications through intra-layer parallelism and inter-layer pipelining. However, balancing these two strategies is a difficult task that demands a flexible architecture to deploy models and an optimization approach, that is, capable of selecting an optimal strategy from an enormous mapping space. This article presents FLIP2M, a holistic solution for mapping multi-model AR/VR workloads on tiled architectures. FLIP2M consists of (1) FLIP, an acceleration fabric that supports a wide variety of optimizations through flexible on-chip communication, and (2) OASIS, an optimization framework based on dynamic and constraint programming, that is, capable of selecting an efficient strategy for mapping multi-model workloads onto FLIP. We demonstrate FLIP2M on an FPGA prototype of FLIP that features 36 accelerators and 7 DDR4 controllers. Using OASIS-generated mappings for three different multi-model AR/VR workloads, FLIP2M achieves up to 1.94× improvement in latency, 1.37× in energy, and 2.59× in energy-delay product relative to a FLIP baseline without intra-layer resource allocation flexibility and inter-layer pipelining.

CCS Concepts: • **Computer systems organization** → **System on a chip**; *Embedded hardware*; *Interconnection architectures*; *Heterogeneous (hybrid) systems*; • **Computing methodologies** → *Mixed/augmented reality*; *Virtual reality;*

Additional Key Words and Phrases: Hardware accelerators, tiled architectures, neural networks, intra-layer parallelism, inter-layer pipelining, AR/VR workloads, FPGA prototyping, design space exploration

ACM Reference Format:
Gabriele Tombesi, Je Yang, Joseph Zuckerman, Davide Giri, William Baisi, and Luca Carloni. 2025. FLIP2M: Flexible Intra-layer Parallelism and Inter-layer Pipelining for Multi-model AR/VR Workloads. *ACM Trans. Embedd. Comput. Syst.* 24, 5s, Article 85 (September 2025), 27 pages. https://doi.org/10.1145/3762656

This work is partially supported by a DOE award (A#: DESC0024458) and a Columbia Center of Artificial Intelligence Technology (CAIT) Award.

Authors' Contact Information: Gabriele Tombesi, Computer Science, Columbia University, New York, NY, USA; e-mail: gtombesi@cs.columbia.edu; Je Yang, Computer Science, Columbia University, New York, NY, USA; e-mail: je.yang@cs.columbia.edu; Joseph Zuckerman, Columbia University, New York, NY, USA; e-mail: jzuck@cs.columbia.edu; Davide Giri, Computer Science, Columbia University, New York, NY, USA; e-mail: davide.giri@columbia.edu; William Baisi, Columbia University, New York, NY, USA; e-mail: wb2426@columbia.edu; Luca Carloni, Computer Science, Columbia University, New York, NY, USA; e-mail: luca@cs.columbia.edu.

ACM 1539-9087/2025/09-ART85
https://doi.org/10.1145/3762656

1 Introduction

Machine Learning (ML) techniques have been rapidly adopted in applications that run on a broad spectrum of computing platforms, from edge devices to data centers. To meet performance and efficiency targets, ML models are increasingly deployed on specialized hardware accelerators.

In **Augmented and Virtual Reality (AR/VR)** applications, multiple **Deep Neural Network (DNN)** tasks—such as object detection for obstacle avoidance, hand tracking and hand-pose estimation for intuitive user inputs—are executed concurrently to enhance the user experience [29, 48].

These multi-model workloads exhibit substantial heterogeneity not only across concurrently executing models but also within individual models; models are composed of a diverse set of layers, each characterized by distinct computational intensity, dataflow patterns, and memory footprint. Moreover, as models continue to evolve, they may require the addition of specialized operators. This pronounced intra-model and inter-model heterogeneity demands flexible hardware architectures that can dynamically allocate resources to efficiently handle the diverse computational demands.

A widely adopted approach to addressing these challenges is to exploit *intra-layer parallelism* through *tiled architectures*, in which multiple accelerator cores—connected via a **network-on-chip (NoC)**—cooperate to process each layer [14, 15]. These architectures can adapt to layers with varying sizes and complexities, thereby reducing idle periods for smaller layers and mitigating data-movement overheads for larger ones. In multi-model AR/VR workloads, intra-layer parallelism enables dynamic resource allocation, such that the number of accelerators can be tailored to the computational intensity of each layer and different dataflow strategies can be leveraged to accommodate the diverse layer types [41, 42].

However, many existing multi-model solutions still process a single layer at a time per model using all available resources, thus overlooking potential gains from *inter-layer pipelining*. By allowing multiple accelerators to process consecutive layers simultaneously, inter-layer pipelining can significantly reduce costly off-chip memory accesses, which become increasingly expensive as more models are executed concurrently.

Implementing both intra-layer parallelism and inter-layer pipelining requires advanced architectural support. Specifically, *groups* of accelerators must be flexibly allocated to exploit intra-layer parallelism, while simultaneously fusing consecutive layers into *segments* to enable inter-layer pipelining. We define a segment as a sequence of adjacent layers that are executed concurrently by separate accelerator groups. This approach requires a robust on-chip interconnect capable of managing complex communication patterns. Moreover, mapping multiple models, each with different sequences of layers, onto such a system introduces a challenging optimization problem that involves determining how to partition each model into segments and how to allocate hardware resources in a manner that maintains high-performance and efficient concurrent execution.

While many existing works exclusively focus on either intra-layer parallelism [16, 21, 24, 25, 28, 50, 53, 55] or inter-layer pipelining [17, 22, 44], only a few consider the integration of these different execution strategies [3, 15, 35]. No prior work offers a holistic solution that uniformly supports arbitrarily sized accelerator groups and segments, while effectively exploring the vast mapping space inherent to multi-model settings. Without a systematic method to explore this space, suboptimal mappings can lead to underutilization of resources, increased latency, and inefficient use of power.

To fill this void, we developed FLIP2M (Figure 1) as a holistic solution that integrates:

— **FLIP** — a tiled acceleration fabric that provides flexibility in the allocation of computational resources and off-chip memory bandwidth, both across concurrently executed layers of different models and along pipelined layers of the same model; FLIP efficiently supports diverse communication patterns through integrated**Peer-to-Peer (P2P)** and multicast mechanisms on a NoC.

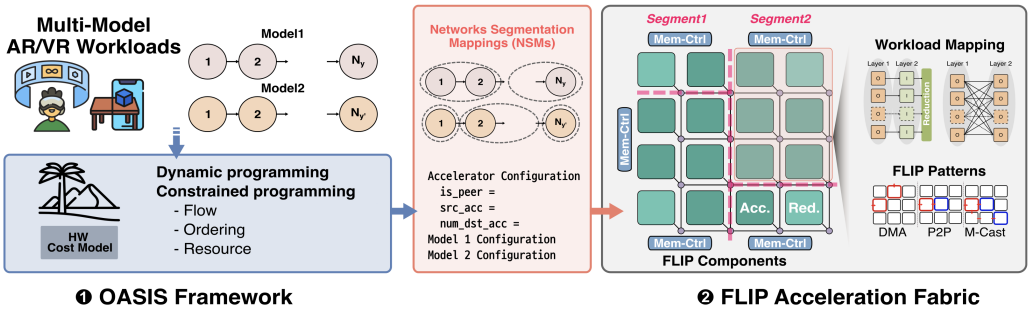

Fig. 1. An overview of FLIP2M, combining the FLIP acceleration fabric with the OASIS optimization framework.

— **OASIS** — an optimization framework, based on dynamic and **constraint programming** (**CP**), that determines how to optimally segment DNNs and map them onto a FLIP instance; OASIS explores the vast space of possible mappings when combining intra-layer and inter-layer execution strategies in multi-model AR/VR workloads.

To evaluate FLIP2M, we realized an **FPGA-based prototype of FLIP** that features 36 accelerators. We used the FLIP prototype to comprehensively evaluate how different OASIS-generated mappings perform under FLIP's flexible resource-allocation schemes when executing three different multi-model workloads for AR/VR from the XRBench benchmark suite [29]. By providing architectural flexibility to combine intra-layer parallelism with inter-layer pipelining, along with an optimization framework to navigate the complex mapping space this flexibility enables, FLIP2M offers a unified and efficient platform capable of meeting the diverse and concurrent demands of modern multi-model AR/VR workloads. Altogether, our experiments show FLIP2M's ability to significantly improve latency (1.94x), energy (1.37x), and **energy-delay product** (**EDP**) (2.59x) when running real multi-model AR/VR workloads.

2 Background

2.1 Heterogeneity of Multi-Model AR/VR Workloads

Modern ML workloads exhibit significant heterogeneity both within individual models and across multiple different models [14, 15, 53]. This diversity reflects varied design goals, such as maximizing accuracy in classification tasks, providing pixel-level segmentation, or supporting robust detection and tracking. Many classification and object-detection networks progressively reduce the spatial dimensions of feature maps (e.g., via pooling or strided convolutions) while increasing the number of channels [14]. Segmentation networks like Unet, in contrast, must eventually restore high-resolution outputs through upsampling or transposed convolutions [15]. Transformer-based models introduce additional heterogeneity, relying on attention mechanisms and feed-forward blocks with large matrix multiplications, layer normalization, and other operators that differ markedly from traditional convolutions [12].

Table 1 summarizes the heterogeneity in both layer configurations and operation requirements of DNN models. The ratio between the activation and weight sizes for a given layer is one way to characterize its shape; for MobileNet-v2, the largest such ratio is ~ 137,000 times larger than the smallest. In addition to standard 2D convolution and matrix-matrix multiplication, these DNNs rely on depth-wise and up-scale convolutions. This pronounced heterogeneity poses significant efficiency challenges in terms of both latency and energy consumption, as prior accelerator solutions are often over-specialized for specific sets of DNN layers.

Table 1. Heterogeneity in DNN Models Used in AR/VR Workloads [48]

Task	Model	Per-Layer Activation-Weight Size Ratio	Layer Operations
Object Detection	MobileNet-v2 [39]	Min: 0.0057, Median: 2.04, and Max: 784	Conv2D, PW-Conv, DW-Conv, and Residual
Object Classification	ResNet-50 [19]	Min: 0.0106, Median: 0.68, and Max: 49	Conv2D, FullyConnected, and Residual
Face Recognition	Vgg16 [40]	Min: 0.043, Median: 1.36, and Max: 87.111	Conv2D, FullyConnected
Keyword Spotting	SqueezeNet [20]	Min: 0.073, Median: 2.64, and Max: 189.06	Conv2D, FullyConnected, and Residual
Hand Tracking	Unet [38]	Min: 0.13, Median: 8.88, and Max: 568	Conv2D, FullyConnected, UpConv, and Concat
QuestionAnswering	MobileBert [43]	Min: 0.125, Median: 0.5, and Max: 8	Matrix-matrix multiplication, and Residual

Layer operations include 2D convolution (Conv2D), point-wise 2D convolution (PW-Conv), depth-wise convolution (DW-Conv), residual block (residual), up-scale convolution (UpConv) and concatenation (Concat).

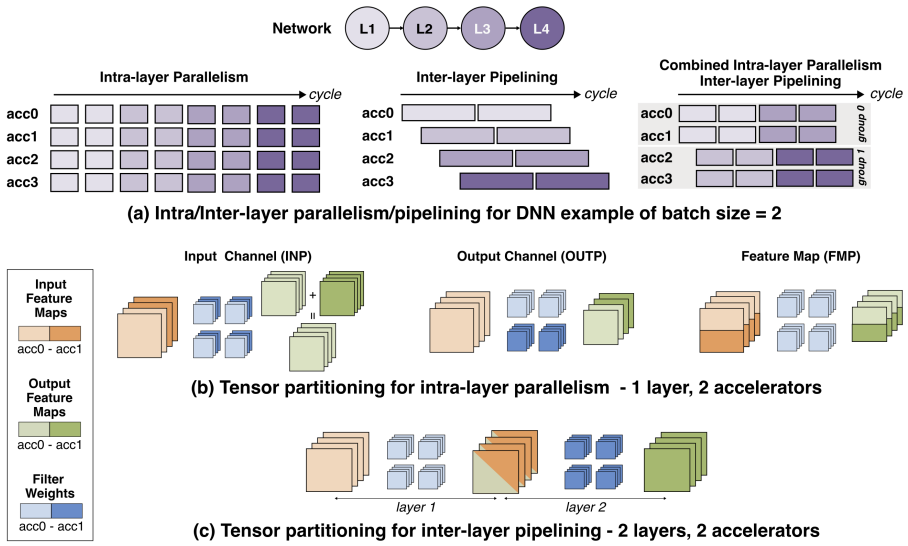

Fig. 2. Intra-layer parallelism and inter-layer pipelining in DNNs.

In multi-model AR/VR workloads, multiple models with distinct layer shapes and operator sets often run simultaneously, further broadening the range of operational requirements and resource demands. This concurrency can lead to contention for on-chip compute resources and off-chip memory bandwidth, complicating the mapping of each layer or operator [15, 53]. Flexible resource allocation and support for diverse operators (e.g., depth-wise convolution, batch normalization, pooling) are thus essential for maintaining high performance and energy efficiency. In general, accommodating both intra-model and inter-model heterogeneity requires hardware designs that can seamlessly adapt to the varying demands of multi-model AR/VR workloads.

2.2 Execution Strategies in DNNs Acceleration

To address the above challenges, a variety of *execution strategies* have been adopted, each with distinct trade-offs in performance and design complexity. Figure 2(a) illustrates these strategies for a four-layer DNN deployed on four accelerators with a batch size of two. Intra-layer parallelism partitions the execution of a given layer across all four accelerators, which compute separate batches sequentially, storing intermediate features off-chip. Inter-layer pipelining, in contrast, dedicates each accelerator to a single layer; although a single accelerator may become a bottleneck for one layer,

the computations of distinct layers and different batches can overlap by forwarding intermediate features on-chip, thereby reducing off-chip memory accesses.

In particular, these strategies can be combined: *groups* of accelerators can each share the workload of one layer while simultaneously forwarding outputs to another group assigned to the subsequent layer. The combined example in Figure 2(a) splits the four accelerators into two groups, each of which execute two layers, to simultaneously parallelize and pipeline the model's execution.

2.2.1 Intra-Layer Parallelism. Intra-layer parallelism focuses on the computational bottlenecks within a single layer by distributing its workload among multiple accelerators. It can be implemented using different schemes. Figure 2(b) shows a classification of *parallelism schemes*, as they apply to a 2D convolution layer, although they can also be applied to other operations.

- **Input Channel Parallelism (INP) :** Each accelerator is responsible for a subset of input channels and the corresponding subsets of filters, generating partial outputs that are accumulated to form the final result.
- **Output Channel Parallelism (OUTP) :** Filters are distributed across accelerators, each of which processes the full set of input channels for its assigned filters, producing complete output feature maps for those filters.
- **Feature Map Parallelism (FMP) :** The spatial dimension of the input feature map is partitioned. Each accelerator processes a distinct spatial region using all the filters; thus, each produces a complete slice of the output feature maps.

By selecting an appropriate parallelism scheme, the accelerators can optimize utilization and match the memory constraints of the system. Significant effort has also been devoted to exploiting fine-grained parallelism within each accelerator through specialized dataflow strategies targeting various forms of data reuse. Heterogeneous dataflow accelerators [28] further extend this approach by integrating multiple, distinct dataflows within a single tiled architecture. By contrast, we focus on combining coarse-grained intra-layer parallelism with inter-layer pipelining, rather than specializing accelerator dataflows for fine-grained parallelism.

2.2.2 Inter-Layer Pipelining. Complementary to intra-layer parallelism, inter-layer pipelining—often termed layer fusion—fuses sequentially dependent layers into a pipeline, which we refer to as a *segment*. For a given DNN, only one segment is active at any given time, and the execution of the layers within the segment can be pipelined by forwarding intermediate activations on-chip rather than storing them off-chip. This can significantly reduce external memory accesses, a benefit that can be magnified when multiple models are run simultaneously. However, the overhead of filling and emptying the pipeline [15] and the additional capacity required in the on-chip buffers [44] can complicate the design. Figure 2(c) shows a simplified example of a two-layer segment executing on two accelerators in pipeline; the output features of the first accelerator are forwarded to the second accelerator as input features. For each batch, input features are generally split into smaller feature chunks that are processed sequentially; this reduces the size of buffers needed to store intermediate data, but diminishes opportunities for fine-grained data reuse within each accelerator [53].

2.2.3 Parallelism/Pipelining Tradeoffs. As shown in Figure 2(a), intra-layer parallelism can be combined with inter-layer pipelining, so that some accelerators process one layer while others concurrently initiate the next. Although this approach can achieve high utilization, balancing these execution strategies is nontrivial. Large layers benefit from intra-layer parallelism, but small layers may underutilize hardware and replicating data (weights, inputs, and partial sums) across multiple cores can increase data-movement overhead. Meanwhile, inter-layer pipelining reduces off-chip memory accesses but inherently limits opportunities for both coarse-grained intra-layer parallelism

(i.e., the number of accelerators dedicated to each layer) and fine-grained data-reuse (i.e, the size of input features used by each accelerator).

3 The FLIP Acceleration Fabric

In this section, we introduce FLIP, an acceleration fabric for tiled architectures that enables flexible intra-layer parallelism and inter-layer pipelining, specifically tailored for multi-model AR/VR workloads. In particular, we describe the architecture of the FLIP acceleration fabric and explain how DNN layers are mapped onto the target architecture.

3.1 The FLIP Architecture

As shown in Figure 1, the FLIP acceleration fabric has a tiled architecture with two main components: compute engines and memory controllers. These components are connected by a NoC, which can scale to support large instances of FLIP with many components, as detailed below.

3.1.1 FLIP Components. The FLIP architecture features two types of compute engines.

Accelerator Tiles implement the core kernels of the target DNN workloads. In the FLIP architecture, these accelerators are *coarse grained* and execute large tasks, such as an entire layer or a significant portion of a layer. In addition to compute logic, the accelerator tiles include their own private buffers. To fill or write back the contents of their buffers, the accelerators issue long load or store bursts on the system interconnect, respectively. As discussed in Section 3.1.2, these loads and stores can be served in several different ways. The accelerator tiles do not need to be implemented in a particular way (i.e., vector lanes, systolic array) or follow a particular programming model; they just must support the key kernels of the target DNN workloads and interface to the FLIP interconnect to support the required communication modes. Our accelerator tiles support comprehensive key kernels listed in Table 1.

Reduction Tiles are dedicated to tensor addition, which is critical for combining partial outputs (e.g., from multiple accelerators executing a shared layer with the INP parallelism scheme of Figure 2) and for merging features from different branches of a residual block, where each branch is executed by one or more accelerators. Providing a dedicated compute engine for this task avoids a costly bottleneck of performing this step from software.

Since both accelerator and reduction tiles perform computations involved in executing portions of a DNN, we broadly refer to them as *compute tiles*.

Memory Controllers. The FLIP architecture supports an arbitrary number of memory controllers, which are used by compute tiles to access off-chip memory (DRAM). This number depends on the constraints of the system and the target workloads. Supporting multiple memory controllers enables scaling the memory bandwidth of the system and mitigates the bottlenecks that can occur due to accelerator contention. Multiple memory controllers also allow for an increase in *isolation* between segments, which is important for our proposed optimization approach. When multiple memory channels are used, the global address space is discretely partitioned among them.

3.1.2 FLIP Communication Patterns. For the efficient mapping of DNN workloads onto the FLIP acceleration fabric, the interconnect must support three different communication patterns. In addition to the support for these three modes, FLIP requires the ability to dynamically switch between their usage.

Direct Memory Access. FLIP uses **direct memory access (DMA)** to load data from the memory controllers into the private buffers of the accelerators. To support efficient transfers of large amounts of data, the NoC must sustain the bandwidth provided by the memory controllers. Model weights are always loaded with DMA, while features can also be loaded or stored with DMA when inter-layer pipelining is not used or when a compute tile is the first or last in a pipeline.

Peer-to-Peer Communication. P2P communication is direct communication between two compute tiles. This is a critical feature for inter-layer pipelining, as it avoids a round-trip to main memory

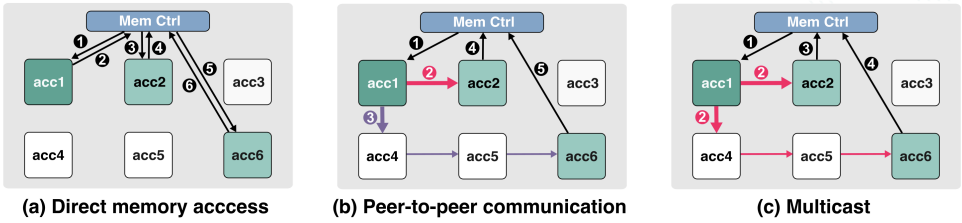

Fig. 3. Data movements for DMA, P2P communication and multicast. The consumer tiles (acc2 and acc6) received data from the producer tile (acc1).

when one compute tile (the *consumer*) uses the output of another (the *producer*) as its input. When using P2P, it is important that the downstream compute tile is ready to consume data, that is, sent by the upstream compute tile; otherwise, data could remain in the NoC, which could result in deadlock. Because of this, P2P requires *synchronization* for data transfers between compute tiles; this can be implemented by the compute tiles themselves, the NoC, or the interface logic between the tiles and NoC.

Multicast Communication. When combining intra-layer parallelism with inter-layer pipelining, it becomes advantageous to support simultaneous communication of data from one compute tile to multiple other tiles. For example, when leveraging OUTP, each compute tile uses the complete set of input features and a partial set of weights to create a partial set of output features. When the compute tiles that process the next layer also operate with OUTP, this partial set must be sent to each downstream compute tile. Multicast communication allows for this with a single transaction on the NoC, avoiding the costly serialization of transfers and the need to either buffer output data locally or store it off-chip.

Figure 3 provides an illustrative example of FLIP's different communication patterns in a scenario where two consumers (*acc2* and *acc6*) process the output of a single producer (*acc1*). When only leveraging DMA, the producer stores the output of its computation off chip, which is then accessed by the consumers sequentially (Figure 3(a)). P2P allows the producer's output to be kept on chip. However, because there are two consumers of *acc1*'s data, it must buffer its entire output to sequentially perform two P2P transactions (Figure 3(b)). Finally, multicast allows the output of *acc1* to be sent to both consumers in parallel (Figure 3(c)).

3.2 Workload Mapping

When mapping DNN layers onto FLIP, we can exploit intra-layer parallelism using one of the three schemes introduced in Section 2.2.1. Each of these schemes provides a way to parallelize the computation of a single layer across different accelerators. They can also be used to temporally split the computation into multiple chunks executed on the same accelerator or group of accelerators. As mentioned in Section 2.2.2, inter-layer pipelining demands reduced input feature sizes to minimize the additional buffering required for storing intermediate features. For this reason, following previous designs [44], we apply FMP temporally, while focusing on INP and OUTP for spatial parallelization. This section illustrates how different on-chip communication patterns emerge when deploying different parallelism schemes, depending on whether we execute one layer in isolation (single-layer segment) or fuse consecutive layers into a segment (multi-layer segment).

3.2.1 Single-Layer Segment. In OUTP, each accelerator is assigned a distinct subset of weights, thus requiring it to load the entire input feature set (via DMA reads) and its assigned weights. It then writes the complete output channel back to memory. Since each accelerator produces a disjoint subset of output channels, no communication is needed across accelerators of the same group. Conversely, in INP, each accelerator processes a portion of the input features, thus generating partial

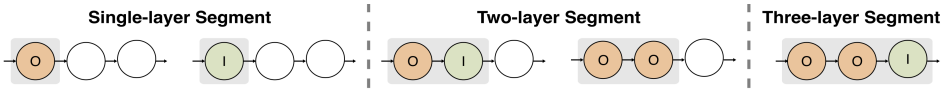

Fig. 4. Possible FLIP segments with varying layer numbers and parallelization schemes (OUTP=*O* and INP=*I*).

outputs for all output channels. These partial outputs are sent to a reduction tile via P2P transfers. The reduction tile performs round-robin accumulation of the partial outputs, and ultimately writes the complete set of output features to external memory.

3.2.2 Multi-Layer Segment. When two or more consecutive layers are combined into a segment using inter-layer pipelining, the accelerators responsible for a layer must retrieve data directly from those handling the previous layer. This approach can significantly reduce both latency and off-chip bandwidth usage, but requires more elaborate on-chip communication patterns.

FLIP provides flexibility in combining parallelism schemes for segments that contain an arbitrary number of layers. However, our observations indicate two key limitations. First, an INP-OUTP configuration—where the first group of accelerators (producers) uses INP and the second group (consumers) uses OUTP—introduces performance bottlenecks in the reduction tile, as it must consolidate partial outputs from the first layer before the second layer can commence. This overhead substantially negates the benefits of pipelining, leading us to omit the INP–OUTP scheme from our analysis. Second, an excessive number of layers within a segment pipeline can degrade overall system performance, mainly due to increased pipeline filling overhead and reduced parallelism opportunities, as fewer resources are available per layer [15]. Consequently, we restrict our analysis to segments comprising a maximum of three layers. Therefore, an invariant of the evaluation framework and all experiments presented in this work is that, at any given time, no more than three layers of each model are simultaneously deployed on the FLIP fabric.

With these restrictions in mind, Figure 4 shows the five types of possible segments supported by FLIP that we consider in the remainder of this work. Specifically, two types of single-layer segments (OUTP and INP), two types of two-layer segments (OUTP–INP and OUTP–OUTP), and one type of three-layer segment (OUTP–OUTP–INP). The following explains how we support the two possible combinations of serial parallelism schemes, namely OUTP-OUTP and OUTP-INP.

— **OUTP–INP:** The first group executes OUTP on the first layer, dividing the output channels among themselves. Meanwhile, the second group employs INP on the second layer, each requiring only a subset of the input channels. The left side of Figure 5(a) shows the simplest case where the number of producers (N_P) is the same as the number of consumers (N_C). In this case, each consumer can receive inputs from the same producer via P2P transfers for the entire duration of the computation. However, if N_P differs from N_C, then P2P load requests must be issued differently. For example, when N_P is greater than N_C, a single consumer must receive multiple subsets of channels from different producers in a round-robin fashion, as shown in the middle of Figure 5(a) Conversely, if N_C exceeds N_P, multiple consumers must receive data from a single producer, as shown on the right side of Figure 5(a)

— **OUTP–OUTP:** In this case, both producers and consumers use OUTP to parallelize the execution of the layers to which they are assigned. Each producer generates a distinct chunk of output channels, but every consumer requires the entire set of input channels. Consequently, multicast transfers are needed. In this case, each producer sends its outputs to all N_C destinations simultaneously, and each consumer receives data from all N_P producers, as shown in Figure 5(b)

3.2.3 Inter-Segment Independence Assumption. In the FLIP architecture, each segment can be allocated to a clustered group of compute engines and memory controllers, as shown on the right

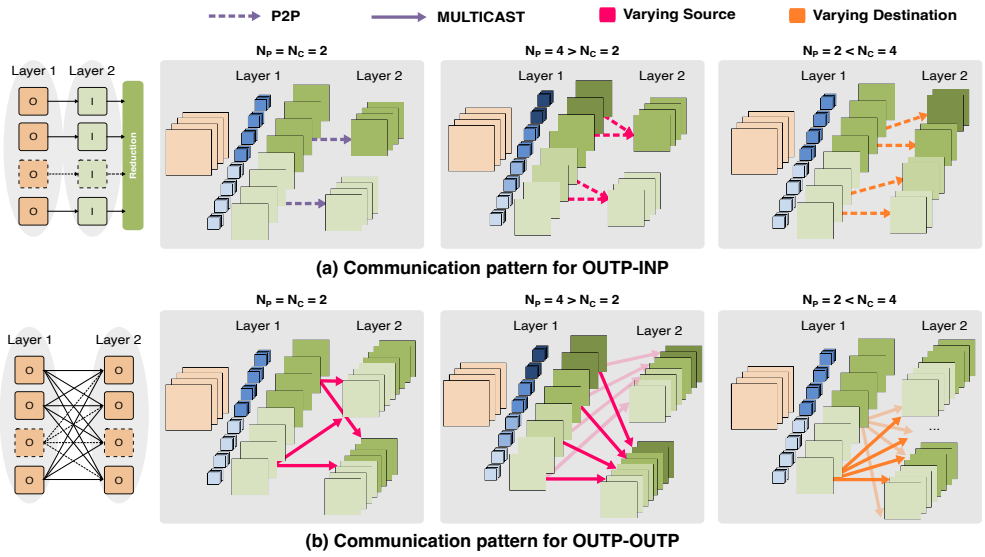

Fig. 5. FLIP communication patterns for a two-layer segment.

side of Figure 1. By confining the on-chip data movement of each segment within its assigned cluster, the NoC-based interconnect ensures minimal interference between concurrently active segments. This clustering strategy, combined with the modularity of tiled architectures, guarantees the *inter-segment independence assumption*, which greatly simplifies the subsequent cost-modeling process: the performance of one segment can be estimated in isolation, without exhaustively analyzing interactions across the entire mesh interconnect. Hence, as FLIP scales to multi-model scenarios, each segment remains independently analyzable, keeping system-wide complexity manageable.

4 The OASIS Optimization Framework

This section presents OASIS, a versatile optimization framework that coordinates network segmentation, resource allocation, and scheduling for multi-model AR/VR applications running on the FLIP acceleration fabric. Section 3 describes how FLIP's architectural features enable flexibility in combining coarse-grained intra-layer parallelism and inter-layer pipelining opportunities. Here, we show how OASIS leverages that flexibility to handle the inherent heterogeneity of multi-model AR/VR workloads in terms of both intra-model and inter-model variability.

The OASIS framework aims to optimize the user-specified optimization objective in the rich mapping space created by the flexible options provided by the FLIP accelerator fabric to exploit parallelism and pipelining. Although we ultimately target multi-model settings, OASIS is designed to adapt to situations where only a single model may be running at a given time [1]. This contrasts with previous work, which often employs the same strategy for both single-model and multi-model applications. In OASIS, we distinguish a simpler single-model solver from a more sophisticated multi-model solver and dynamically select the solver according to the number of active models.

[1](1) Many production pipelines still dedicate an entire accelerator to a single DNN, so a single-model solver remains directly useful to practitioners. (2) Even inside a multi-model graph, execution often collapses into linear chains of layers from a single network (for example, in case of inter-model functional dependencies).

4.1 Inputs and Mapping Space Definition

OASIS takes two primary inputs: *DNN model topologies* and the target *FLIP prototype*. For each DNN, we assume a model described in an ONNX-style format, including per-layer shapes, operators (e.g., 2D convolutions, depth-wise convolutions, and up-scale convolutions), and the network's connectivity (e.g., residual blocks, concatenations). OASIS is also able to handle static functional dependencies across the specified input models. The user encodes any required relations directly in the workload description fed to the solver. Once these dependencies are added, the solver's constraint set expands automatically and the mapping/schedule it returns is consistent with the new topology. The FLIP prototype description includes the number of accelerator tiles, reduction tiles, and memory controllers available on the SoC. We assume that a single reduction tile is available for partial-output summations and residual blocks for each model.

As mentioned in Section 3, a *segment* is a fused stack of consecutive layers of a DNN. Although OASIS is implemented to support any such segment, we limit our discussion to the five types of segments of Figure 4. We formally define a *segment type (ST)* by two characteristics: the *segment depth (sd)* —the number of layers in the segment—and the *segment parallelism* ($\overline{sp} = \{sp_1,...,sp_{sd}\}$)—the list of intra-layer parallelism schemes that each layer of the segment uses. Each layer i of a DNN can potentially be the start layer of a segment with a given *ST*. For some layers, the segments originating from them can cover only a subset of all the supported segment types, depending on the network topology (e.g., a layer in a residual branch might preclude segments with $sd > 1$, while the second-to-last layer of a network cannot start segments with $sd > 2$).

For each segment with a given *ST=(sd,* $\overline{sp}$*)*, we define a *segment resource SR* = ($\overline{n}_{acc} = \{n_{acc,1},...,n_{acc,sd}\}, n_{mem}$), which determines how many accelerator tiles each layer in the segment uses and how many memory controllers the whole segment uses. Consider a FLIP instance with R accelerator tiles and M memory controllers. Each layer in each segment can use a power-of-two number of accelerators from {1,2,4,...,R}—provided the total number of accelerators across all layers in all active segments does not exceed R – and each segment can choose from {1,2,...,M} memory controllers—provided that the total number of memory controllers across active segments does not exceed M. Given a network with N layers and a layer i of the network, we define a *segment mapping* sm_i as the execution of a segment originating from layer i with segment type *ST*, using a segment resource *SR*. We define the *space of segment mappings* originating from layer i as :

$$\mathcal{SM}(i) = (\mathrm{ST} = (sd, \overline{sp}), \quad \mathrm{SR} = (\overline{n}_{acc}, n_{\mathrm{mem}})), \tag{1}$$

with $sd \in \{1,2,3\}$; $sp_k \in \{O,I\}$, $n_{acc,k} \in \{1,2,..,R\}$, $\mathrm{k} \in [1,sd]$; $n_{mem} \in \{1,2,..,M\}$.
Given a network y, a *network mapping* is a candidate solution to map the network on the available FLIP resources and is defined as a sequence of P segment mappings that includes all layers of y:

$$network\ mapping = (sm_{i_1}, sm_{i_2}, ..., sm_{i_P}) \in \mathcal{SM}(i_1) \times \mathcal{SM}(i_2) \times \cdots \times \mathcal{SM}(i_P), \tag{2}$$

where sm_{i_j} is the *jth* segment mapping of the sequence that starts at layer i_j and has segment depth sd_j, such that the following conditions hold:

$$i_1 = 1, \quad i_{j+1} = i_j + sd_j, \quad \sum_{j=1}^{P} sd_j = N. \tag{3}$$

Given a description of the input models and available hardware resources, the goal of OASIS is to find the optimal network mapping from the *space of network mappings* ($\mathcal{NM}$) for each model, selecting the sequence of segment mappings that minimizes the cost across all models. This consists of finding both the optimal partition of the input networks into segments and the optimal allocation of accelerators, memory controllers, and parallelism schemes for each layer in each segment mapping. The notation used throughout this section is summarized in Table 2.

Table 2. Notation used in OASIS Formulation

Notation	Description
N	Number of layers for single-model scenario
N_y	Number of layers for network y for multi-model scenario
R	Total number of accelerators
M	Total number of memory controllers
ST (Segment Type)	ST = $(sd, \overline{sp})$
sd (Segment Depth)	Number of layers in the segment
$\overline{sp}$ (Segment Parallelism)	$(sp_1,..,sp_{sd})$ = List of intra-layer parallelism scheme for each layer of the segment
SR (Segment Resource)	SR = $(\overline{n}_{acc}, n_{mem})$
$\overline{n}_{acc}$	$(n_{acc,1},..,n_{acc,sd})$ = List of number of accelerators used by each layer of the segment
n_{mem}	Number of the DDRs used by each segment
sm_i (Segment Mapping)	Mapping of a segment originating from layer i
network mapping = $(sm_1, sm_{i_2},..,sm_{i_p})$	Sequence of segment mappings, partitioning the entire network y
$\mathcal{SM}(i)$ (sm-space for layer i)	(ST,SR) = Space of segment mappings originating from layer i
$\mathcal{NM}(y)$ (network mapping space for network y)	Space of network mappings for network y

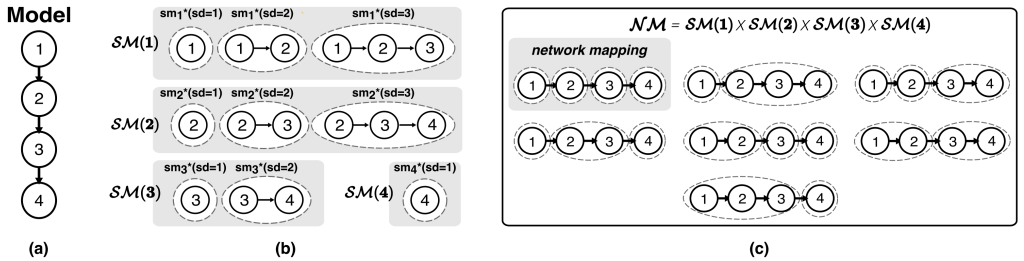

Fig. 6. Space of segment mappings and network mappings for a four-layer network, with fixed $\overline{sp}, \overline{n}_{acc}, n_{mem}$.

An example of a four-layer network is shown in Figure 6(a), Figure 6(b) illustrates the space of segment mappings ($\mathcal{SM}(i)$) for each layer i, and Figure 6(c) shows the corresponding space of network mappings ($\mathcal{NM}$). For simplicity, in this example, we assume fixed parallelism schemes and resources ($\overline{sp}, \overline{n}_{acc}, n_{\text{mem}}$). However, when considering these additional knobs, both $\mathcal{SM}(i)$ and $\mathcal{NM}$ would encompass a broader set of possible mappings than the ones presented.

4.2 Cost Model

Because OASIS's goal is to handle the vast mapping space unleashed by FLIP's flexibility, a robust cost model is essential to estimate the performance of each segment mapping. Crucially, the inter-segment independence assumption, enabled by clustering resources at the architectural level, makes the cost model tractable for multi-model workloads. Whether using an in-house profiler or external frameworks [27, 31, 37], OASIS users tanot ergeting FLIP benefit from this simplified perspective, as it enables independent estimation of each segment mappings's execution cost. This eliminates the need to model complex cross-segment interference on the NoC, which would otherwise render the search space prohibitively large. We refer to the target objective cost and latency for a segment mapping sm_i, as estimated by the cost model, as c_{sm_i} and d_{sm_i}, respectively. The specific cost c that we seek to minimize can be chosen by the user; no matter the chosen cost, the latency is required, as Section 4.4 will describe.

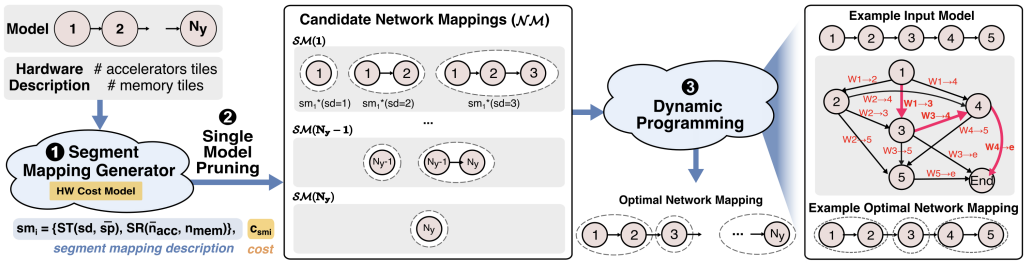

Fig. 7. OASIS's single model solver.

4.3 Single-Model Solver

In this section, we describe our single-model solver, as illustrated in Figure 7, including the mathematical formulation of its mapping space and our dynamic programming approach to find the optimal network mapping. We extend this solver for multi-model deployments in Section 4.4.

As mentioned above, the space of possible segment mappings starting from each layer of a N-layer network ($\mathcal{SM}(i), 1 \leq i \leq N$) is derived from combining multiple independent knobs (sd, $\overline{sp}$, $\overline{n}_{acc}$, n_{mem}). A *segment mapping generator* is used to generate all these configurations for a given input model and FLIP prototype description ❶.

Under a single-model deployment assumption, where only one model is active at a time, each segment, with segment type $ST(sd, \overline{sp})$, can fully utilize all the resources of the FLIP acceleration fabric. Although the parameters of ST and SR collectively determine the cost of executing a given segment mapping, the segment depth itself (sd) is the only one that inherently governs how much progress is made in the execution of the network. These observations allow us to apply a pruning step ❷ across the parallelism and resource allocation knobs by choosing, for each allowed sd, the combination of ($\overline{sp}$, $\overline{n}_{acc}$, n_{mem}) that minimizes the objective cost. Consequently, each layer retains only one optimal segment mapping for each sd, similarly to the example shown in Figure 6. This is because the optimal sd cannot be defined a priori, but must be determined in the context of the execution of the entire network. In our formulation, if a layer i can start segments with possible $sd = 1,2,3$, this pruning process reduces $\mathcal{SM}(i)$ to at most three segment mappings, one for each sd, thereby substantially shrinking the search space of the solver.

Even after pruning, the space of network mappings ($\mathcal{NM}$) can remain impractical to explore. Consider the simplified case of a DNN with layers organized with a linear dependence, i.e., without any residual blocks. Although there are five possible ST (Figure 4) and a larger amount of possible SR allocations, the pruning step reduces the number of possible mappings to three per layer. By defining $T(N)$ as the size of $\mathcal{NM}$ for a N-layer network with the characteristics specified above, we can infer the following linear recurrence :

$$T(N) = T(N-1) + T(N-2) + T(N-3) \quad where \;\; T(0) = 1, \;\; T(1) = 1, \;\; T(2) = 2. \tag{4}$$

Solving the characteristic equation of the Tribonacci recurrence of Eequation 4 yields $T(N) = \Theta\big((1.8393)^N\big)$, which means that the mapping space grows exponentially with the network depth N, at a per-layer growth factor of about 1.84. For a moderate value of $N = 30$, we get $T(30) = 53{,}798{,}080$. For N=50, the size of $\mathcal{NM}$ becomes prohibitively high. Thus, even after pruning, exploring all the network mappings remains infeasible even for moderate values of N. Hence, we introduce a dynamic-programming–based solver to efficiently find the optimal network mapping ❸. We represent the network as a **directed acyclic graph (DAG)** where each node corresponds to one layer. An edge from layer i to layer g represents the execution of a segment mapping sm_i, and thus is assigned a weight equal to the cost of it. For a single-layer segment from Layer 1 to Layer 2, the weight matches the cost of the optimal sm_1 with $sd = 1$; similarly, for a three-layer segment from

Layer 2 to 5, the weight is the cost of the optimal sm_2 with $sd = 3$. Formally, we define the weight associated with an edge from layer i to layer g as :

$$W(i \to g) = c_{sm_i^*(sd=g-i+1)}. \tag{5}$$

Where $sm_i^*(sd = g-i+1)$ is the optimal segment mapping starting from layer i and executing up to layer g, i.e., with $sd = g-i+1$. An optimal network mapping is found by minimizing the cumulative cost across all arcs from Layer 1 up to Layer N+1:

$$C(i) = \begin{cases} \min\limits_{g>i} \left[W(i \to g) + C(g) \right] & 1 \le i \le N, \\ 0, & i = N+1 \end{cases}, \tag{6}$$

where $C(i)$ is the minimal total cost from layer i onward. Solving this recurrence via dynamic programming is equivalent to finding the shortest path in the weighted DAG shown in Figure 7. This approach yields the minimal-cost network mapping with a complexity of $O(N \cdot E)$, where N is the number of layers in the network, and E stands for the number of edges leaving a node of the DAG, which in our setting varies from 1 to the number of allowed segment depth values, i.e., $1 \le E \le 3$.

In the following subsection, we extend this methodology to the multi-model setting, where multiple concurrent models must share FLIP's accelerators and off-chip DDR accesses simultaneously.

4.4 Multi-Model Solver

In a multi-model deployment scenario, multiple models compete for computational resources and off-chip memory bandwidth. The direct consequence of this is that the assumption considered for the single-model solver is no longer valid, and the pruning step cannot be applied: for each layer, all possible segment mappings need to be considered when searching for an optimal solution, even if they are suboptimal when considered in isolation. As a result, each layer may have up to hundreds of possible segment mappings, thereby making any approach based on dynamic programming infeasible for any network with even moderate values of N. For Y independent networks, each composed of N layers, mapped onto a FLIP instance with R accelerator tiles and M memory controllers, the size of the joint space of network mappings, under the simplified assumption of independence in resource allocation across networks, follows a weighted Tribonacci recurrence, yielding $\Theta\left((\lfloor \log_2 R \rfloor \lfloor \log_2 M \rfloor)^{NY}\right)$ static configurations. Scheduling any fixed configuration is a two-resource multi-mode **Resource-Constrained Project Scheduling Problem** (**RCPSP**), that is, strongly NP-hard [45]; its feasible-schedule set already contains at least $2^{\Theta(NY)}$ elements. Consequently, the combined mapping and scheduling space scales as $\Omega\left(\exp(\Theta(NY))\right)$,[2] i.e., it is exponential in the problem size, which rules out exhaustive enumeration.

To extend our solver to support multiple DNNs running concurrently on a single FLIP instance, we adopt a multi-level decision problem formulation, similar to the approach in [21, 28, 35]. We show the overview of our proposed approach in Figure 8. After generating all possible segment mappings for each model ❶, we use a *Network Partition Engine* ❷ to split each model into smaller *windows* by grouping consecutive layers such that each window encompasses roughly the same number of **multiply-accumulate** (**MAC**) operations. This first partitioning step greatly reduces the search space by ensuring that each window is more tractable. Because the models may differ in depth and in layer-wise MAC distribution, some of the later windows can end up holding layers from one model only. In these cases, we simply treat these windows with the lighter single-model solver, which reaches an optimal solution quicker for this restricted sub-problem. Otherwise, we feed the multi-model window

[2] $|\Omega_{\text{total}}| = \Omega\left((\lfloor \log_2 R \rfloor \lfloor \log_2 M \rfloor)^{NY} 2^{\alpha NY}\right) = \Omega\left((AB)^{NY} 2^{\alpha NY}\right) = \Omega\left(AB2^{\alpha}\right)^{NY} = \Omega\left(\exp\left(NY \ln(AB2^{\alpha})\right)\right) = \Omega\left(exp\left(\Theta(NY)\right)\right)$

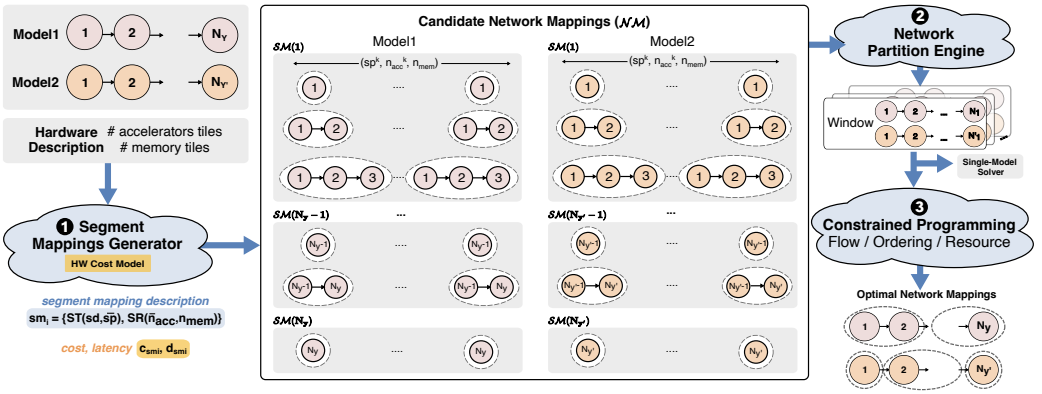

Fig. 8. OASIS's multi model solver.

into a CP solver enhanced with SAT (Boolean satisfiability) solving to handle large combinatorial problems efficiently ❸. This hybrid method combines classical CP techniques with conflict-driven clause learning from SAT, greatly improving efficiency on large combinatorial problems. Next, we describe the *variables* and *constraints* for the CP-based formulation of our window optimization problem. [3]

Decision Variables. For each model y, each layer i in the model, and each segment mapping sm_i that can legally start at layer i, we introduce a binary selection variable $x_{y,sm_i} \in \{0,1\}$ indicating whether or not sm_i is selected. As shown in Equation (1), each segment mapping sm_i specifies: segment depth (sd), accelerator tiles (sum of the accelerator tiles $\overline{n}_{acc}$ used across all the layers of the segment) and memory controllers used in the segment (n_{mem}). In addition, the segment mapping is associated with a cost c_{y,sm_i} and latency d_{y,sm_i}, obtained from the cost model. When a segment mapping is chosen, it occupies an interval in time. We assign a start time scheduling variable, $t_{y,sm_i} \in [0,H]$ where $-H$ is a sufficiently large scheduling horizon. The segment then finishes at $t_{y,sm_i} + d_{y,sm_i}$.

Flow Constraints per Network. We ensure that the segment mappings from each model partition the entire network. Mathematically, if $SegEnd(v)$ is the set of all segment mappings that finish at layer v, we enforce for any candidate network mapping that exactly one segment mapping leaves the first layer of each network and exactly one segment mapping ends at the terminal layer (Equation (7)). For each intermediate layer v, the number of segments ending at v must match the number of segments starting at v; this value can either be 1 if the layer is the first layer in a segment or 0, otherwise (Equation (8)).

$$\sum_{sm_1} x_{y,sm_1} = 1, \qquad \sum_{sm_i \in SegEnd(N_y)} x_{y,sm_i} = 1, \tag{7}$$

$$\sum_{sm_i \in SegEnd(v)} x_{y,sm_i} = \sum_{sm_v} x_{y,sm_v} \leq 1. \tag{8}$$

Precedence Scheduling Constraints. Within each model y, the chosen segment mappings must not overlap in time in an inconsistent order. Formally, if a segment mapping sm_i, covering l layers from i to $i+l$, is followed by another segment sm_{i+l}, then:

$$t_{y,sm_i} + d_{y,sm_i} \leq t_{y,sm_{i+l}} \quad \text{(only if } x_{y,sm_i} = 1 \text{ and } x_{y,sm_{i+l}} = 1\text{)}. \tag{9}$$

This constraint ensures that the order of the segments in each model is preserved.

Cumulative Resource Constraints. Considering a FLIP prototype with R accelerator tiles, and M memory controllers, if a segment mapping sm_i, starting at layer i of model y, is active in the interval $[t_{y,sm_i}, t_{y,sm_i} + d_{y,sm_i})$ and requires r_{y,sm_i} accelerator tiles, then the sum of r_{y,sm_i} across all concurrently active segment mappings cannot exceed R (Equation (10)). Similarly, if a segment mapping sm_i demands m_{y,sm_i} memory controllers, the total memory usage of active segments should not exceed

[3] For simplicity, in this description we refer to the portion of each network allocated in each window as the totality of the network.

M (Equation (11)).

$$\forall t \in [0,H] : \sum_{y,sm_i} (r_{y,sm_i} \cdot x_{y,sm_i}) \cdot \mathbf{1}\{t_{y,sm_i} \leq t \leq t_{y,i,sm_i} + d_{y,sm_i}\} \leq R, \tag{10}$$

$$\forall t \in [0,H] : \sum_{y,sm_i} (m_{y,sm_i} \cdot x_{y,sm_i}) \cdot \mathbf{1}\{t_{y,sm_i} \leq t \leq t_{y,sm_i} + d_{y,sm_i}\} \leq M. \tag{11}$$

We model these additional constraints using cumulative constraints, natively supported by our CP-solver.

Objective function. We aim to minimize the overall cost across all models, i.e., the sum of costs for each selected mode. If c_{y,sm_i} is the cost of the selected segment mapping sm_i, starting at layer i of network y, then:

$$Minimize \sum_{y} \sum_{i,sm_i} c_{y,sm_i} x_{y,sm_i}. \tag{12}$$

5 FLIP Prototype

In this section, we present our prototype of the FLIP accelerator fabric, which is based on the open-source ESP platform for **system-on-chip** (**SoC**) design [6, 32]. We first present our implementation of the FLIP accelerator tile, and then discuss its integration into a 49-tile SoC designed using ESP. The SoC is implemented on an FPGA, which provides the basis for developing a cost model for OASIS and also serves as our evaluation platform for the results presented in Section 6.

5.1 FLIP Accelerator

In this section, we describe the internal microarchitecture of the accelerator deployed in our FLIP prototype. The accelerator was designed in SystemC using the MatchLib library [23], which is based on latency-insensitive design [5], and synthesized using Catapult-HLS, operating with 8-bit integer precision. To optimize data reuse for both weights and input features, we target a multi-level dataflow strategy by extending the weight-stationary–local-output-stationary dataflow proposed by Venkatesan et al. [46]. In our design, weight and input feature reuse are exploited at different granularities to minimize the on-chip buffer footprint and reduce off-chip memory accesses. For weight reuse, the accelerator loads a chunk of weights into a dedicated weight buffer and applies them across all *input feature maps (ifmaps)*, which are loaded in multiple chunks; once it has been used with all ifmaps, the current weight chunk is replaced by a new one.

Figure 9 shows the detailed microarchitecture of FLIP accelerator tile. The core computational engine is a **Vector eXecution unit** (**VXE**) composed of multiple lanes of 8 MAC units. In our implementation, the number of lanes is fixed at four, yielding a total of 32 MAC units per accelerator. The weight buffer is provisioned with sufficient read ports to sustain the full throughput of the MAC array, allowing multiple weight values to be read concurrently into a weight collector, which is tightly coupled with the VXE, thanks to the regularity of the access pattern. Accessing input features poses a greater challenge because parallel access to ifmaps would require duplicating data across multiple banks. To address this, a **Patch EXtractor** (**PEX**) serially reads the required elements from the ifmap buffer into an input collector. The PEX also applies the appropriate border-handling methods based on the specified padding configuration. To further reduce the overhead associated with sequential ifmap accesses, our design reuses ifmaps with a finer granularity by convolving the collected input data with the corresponding k weight values over the next k clock cycles.

The accelerator implements the logic required for in-place functions such as batch normalization, ReLU activation, and pooling within a dedicated **Special Function Unit** (**SFU**). In addition, the accelerator is equipped with a peer communication unit, which provides an additional buffering capacity required in P2P/multicast scenarios. This buffer retains a portion of the output data so that downstream

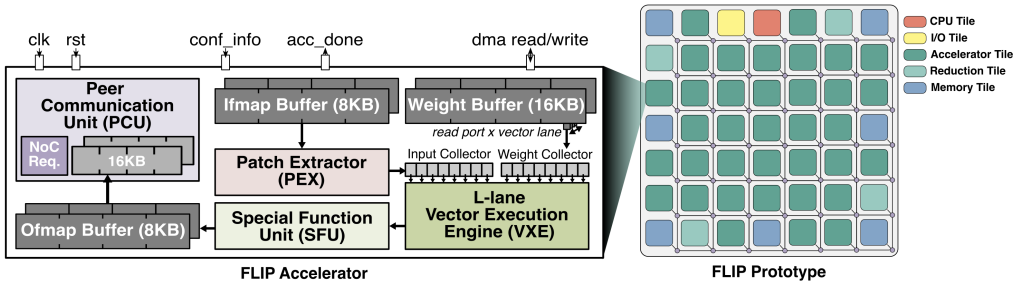

Fig. 9. Detailed implementation of the FLIP prototype.

consumer accelerators can access their inputs as many times as necessary, according to the layer configuration. Furthermore, this unit contains the logic to correctly annotate NoC requests with the source (for a read) and number of consumers (for a write) for each transaction, ensuring compliance with the communication protocols of the FLIP architecture. We intentionally keep all micro-architectural hyper-parameters—such as private-buffer capacity, vector-lane width, and NoC bandwidth—fixed because the goal of this work is *not* a full design-space exploration of the accelerator itself, but rather the mapping/scheduling problem that emerges *after* on-chip resources are set. Freezing the hardware lets us isolate the contribution of our solver over the three combinatorial dimensions it controls: (i) how each layer is partitioned across accelerators (intra-layer parallelism), (ii) how deeply layers are pipelined (inter-layer pipelining), and (iii) how resources are shared among concurrent models.[4]

5.2 FPGA Prototype

We integrated the FLIP accelerator into ESP and designed the FLIP prototype shown on the right side of Figure 5.2. This specific FLIP prototype takes on the architecture provided by ESP for SoC design; we emphasize that FLIP can take on a different implementation.

In addition to compute tiles, an ESP SoC requires a few additional tile types. First, we included a *processor tile* with the CVA6 core [52] to serve as the host of the system. We use the CVA6 core to manage the execution of workloads across compute tiles, i.e., configuring compute tiles according to the workload and mapping strategy. ESP SoCs are *self-hosting*; however, if FLIP were implemented as a PCIe-attached accelerator, for example, it would not need a processor tile. Second, an ESP SoC requires an *auxiliary tile*, which has various peripherals and miscellaneous components. Third, memory controllers in ESP are encapsulated by memory tiles; in our prototype, we included 7 memory tiles (each with a DDR4 controller), which is the maximum number supported by ESP for our chosen FPGA board, the Xilinx XCVU19P. Since we chose a 7x7 tile architecture (based on resource constraints of the FPGA board), our prototype can dedicate the remaining 40 tiles to implement FLIP compute tiles: 36 accelerator tiles and 4 reduction tiles. The reduction tile is designed and synthesized same as the accelerator tile, with internal buffer and logic for tensor accumulation.

The 49 tiles are connected by the ESP NoC, which instantiates multiple networks [51]. Two networks are used by the compute tiles for data transfers; this is necessary to avoid protocol deadlock. ESP natively supports P2P communication between compute tiles. The synchronization is implemented with a consumer-initiated request. When a producer is ready to send the data, it will wait until it receives a request from the consumer, thereby guaranteeing data consumption.

[4]Extending the framework with an outer design-time loop that co-optimizes buffer sizes, NoC bandwidths, or accelerator dataflows remains a promising direction for future work and can reuse the same cost-model interface and RCPSP formulation presented here.

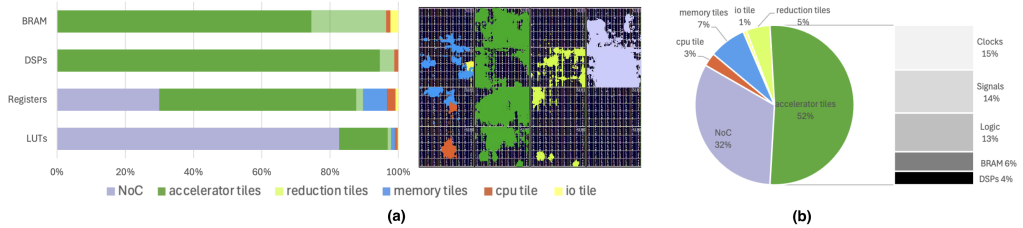

Fig. 10. (a) Resources breakdown for FLIP prototype (b) Power breakdown for FLIP prototype.

However, we needed to make several modifications to ESP to support the desired communication patterns of FLIP. First, the existing P2P implementation requires that the producer and consumer have the same access patterns (i.e., number of transactions and size of each transaction). Because consecutive layers may have different attributes or be parallelized on a different number of accelerators, consumer and producer accelerators will likely have different access patterns, which severely limits the ability to use P2P for FLIP. We relaxed the existing constraint and allowed accelerators to have different access patterns, as long as they agree upon the total amount of data exchanged. Second, we extended the capabilities of the ESP NoC to support multicast communication; multicast communication follows the same dimension-ordered routing protocol in ESP's native NoC. We extended the NoC packets to carry an array of destinations instead of a single one and modified the crossbar to forward packets in multiple different directions, when necessary. Finally, ESP accelerators could previously only have a fixed communication pattern for the entirety of a task. As mentioned in Section 3.1.2, FLIP requires the ability to dynamically switch between its three communication patterns and also dynamically vary the target (source or destination) of requests. We modified the ESP DMA controller to support this capability, and the accelerator can dynamically adjust its communication patterns based on the type of transfer required. For example, weights can be loaded via DMA, while input features can be obtained with P2P, and output features can be sent with multicast communication.

Figure 10(a) summarizes the resource utilization and floor-plan of the FLIP SoC prototype implemented on the XCVU19P FPGA board. Overall, the prototype occupies 47.6 % of LUTs, 21.3 % of registers, 59.6 % of DSPs, and 74.4 % of BRAMs. The majority of on-chip memory is used by the private buffers inside accelerator and reduction tiles. The use of DSPs is concentrated in accelerator tiles, where they implement the SFU operators. LUT and registers are mainly consumed by the accelerator tiles and the NoC logic. The modifications to ESP described earlier in this section to support the FLIP communication patterns incur overheads of 2.2 % of LUTs and 0.2% of registers, compared to a baseline prototype without support for inter-layer pipelining. Figure 10(b) also reports the on-chip power breakdown obtained from post-implementation Vivado power estimates. The largest contributors are accelerator tiles (0.1 W each), reduction tiles (0.009 W each) and the NoC (1.96 W). The dominance of these components aligns with their central role in executing the AR/VR inference workloads mapped onto the prototype.

5.3 FPGA Cost Model and Spatial Allocation Policy

We use our FLIP prototype to profile the space of segment mappings for the networks that are discussed in Section 6. To guarantee the inter-segment independence assumption, we implement a dedicated *spatial allocation policy* that clusters each segment's assigned accelerators tiles and memory controllers in contiguous regions of the SoC. First, when a segment mapping requires a given number of memory controllers (n_{mem}), these tiles are selected from adjacent locations

along the SoC's perimeter to ensure tight spatial proximity among the memory controllers. Then, the specified number of accelerator tiles is allocated by selecting the cluster that minimizes the combined Manhattan distance to the chosen memory tiles. By enforcing a spatial allocation strategy that isolates segment resources, we support a single-segment cost model, that is, robust enough to underpin both single-model and multi-model segment mapping.

We collect a dataset of 22,000 samples, covering the space of segment mappings for all the networks in our benchmarks. Without leveraging the inter-segment independence assumption implemented with our spatial policy, the dataset collection would require a much more extensive effort. Modeling deployment scenarios where concurrently active segments share NoC resources to exchange data among the accelerators is an extremely complex task, that is, beyond the scope of this work.

5.4 Compiler Support for Prototype Execution

The FLIP prototype incorporates a lightweight compiler that translates the optimal network mapping solution generated by OASIS into executable instructions for the hardware. Starting from the high-level parameters already embedded in each segment mapping—such as feature dimensions, channel counts, and activation functions—the compiler augments this information with prototype-specific details. In particular, it examines the spatial organization of accelerator tiles and memory controllers, then binds every segment in the OASIS solution to a concrete set of those resources following the spatial allocation policy specified in Section 5.3. Once a segment has been assigned to its physical tiles, the compiler derives a complete set of communication descriptors: off-chip access patterns are chosen to balance traffic across the memory controllers of the segment, while minimizing data duplication and conforming to the model's off-chip memory layout, and on-chip routes are defined to enable peer-to-peer or multicast transfers that match the segment's parallelism strategy—for example, the round-robin multicast illustrated in Figure 5(b). Finally, the compiler combines the kernel parameters, resource bindings, and communication descriptors into a sequence of accelerator-invocation instructions. In our prototype, these instructions are encoded as configuration tokens, which the ESP CVA6 core transmits to the accelerators' configuration interfaces; other FLIP implementations may adopt different encoding or transport mechanisms, but the compilation flow remains unchanged.

6 Evaluation

6.1 Experimental Setup

Our experimental evaluation of FLIP2M is based on the prototype introduced in Section 5. We first present our multi-model AR/VR benchmark, derived from XRbench [29], which encompasses three distinct multi-model workloads. As summarized in Table 3, each scenario involves a unique combination of models and batch sizes to reflect different processing rates across different AR/VR tasks. We then divide our experiments into two main categories: (1) single-model deployments, where each model is evaluated independently with the full resources of the prototype; and (2) multi-model deployments, where multiple models are executed concurrently under varying resource constraints.

For each experiment, we collect both latency and energy metrics, and infer the EDP. We record execution start times through performance counters instantiated in the processor tile and detect the completion times with the same counters while monitoring the interrupts raised by each accelerator. The energy values were derived from the Vivado power reports by integrating both static and dynamic power over the measured execution times of our benchmark suite. We run the solver on a desktop with an Intel i7-8700K CPU.

6.2 Single-Model Performance

We evaluate three deployment configurations for each model, progressively enabling FLIP2M's capabilities. In the *Baseline*, each model runs on all available accelerators ($n_{acc} = 36$), with a fixed number of

Table 3. FLIP2M's Experimental Multi-model Workload Scenarios for AR/VR use Cases Inspired by MLPerf [48] and XRBench [29]

AR/VR1			AR/VR2			AR/VR3		
Task	Model	Batch size	Task	Model	Batch size	Task	Model	Batch size
Gaze Estimation	ResNet-18	2	Hand Tracking	ResNet-34	4	Eye Segmentation	Vgg16	2
Keyword Detection	SqueezeNet	1	Plane Detection	Unet	2	Depth Refinement	MobileNet-v2	2
Object Detection	MobileNet-v2	2	Speech Recognition	MobileBert	1	Action Segmentation	ResNet-18	4
			Depth Estimation	ResNet-50	2	Speech Recognition	MobileBert	1

We use sequence length = 64 for MobileBert.

memory controllers ($n_{mem} = 4$) and no inter-layer pipelining ($sd = 1$). *FLIP2M-FlexIntra* introduces flexibility in selecting intra-layer parallelism strategies and the number of accelerator tiles and memory controllers, thereby tailoring compute resources and memory bandwidth to each layer's requirements, while *FLIP2M-Full* further enables inter-layer pipelining. We run these three configurations on a single batch of each benchmark network, specifying three different optimization objectives—latency, energy, and EDP—for the OASIS single-model solver. Figure 11 summarizes the results.

For each objective search, we plot the objective metric as well as the other metrics in the same column. Hence, Figure 11 can be analyzed by columns and rows. By column, we can observe the behavior of three metrics when only one of them is optimized in the given search. By row, we can observe the behavior of the same metric when it is the subject of the search versus when it is not. For the latter, the reported values are normalized with respect to those obtained when the metric is the subject of the search. For example, in the case of the energy metric, the plots in Figure 11(d) and Figure 11(f) are normalized to the Baseline in Figure 11(e), which reports the value obtained for the energy search.

The results show that enabling FLIP2M features improves performance for most networks. When minimizing latency, inter-layer pipelining yields a pronounced benefit for seven of the nine models (Figure 11(a)), while also providing secondary energy and EDP improvements (11(d) and 11(g)), as reduced idle cycles and fewer off-chip transfers lower the overall execution cost. On the other hand, optimizing for energy primarily depends on careful intra-layer resource allocation. Consequently, FLIP2M-Full and FLIP2M-FlexIntra often exhibit similar energy consumption (11(e)). For large networks such as ResNet-50 and Vgg16, using fewer accelerators to reduce dynamic power can increase latency, although FLIP2M-Full remains competitive with the Baseline (11(b)). Unet shows minimal gains in latency, due to its memory-bound structure, where frequent off-chip transfers dominate. On average, FLIP2M achieves 1.30×, 2.67×, and 2.71× improvements in latency, energy, and EDP, respectively, when optimizing for each particular objective.

To validate FLIP2M's scalability in single-model deployments, we compare FLIP2M-Full with Tangram [15], a state-of-the-art solution to combine intra-layer parallelism and inter-layer pipelining in tiled architectures. We perform our analysis by varying the number of accelerator tiles (1 to 16) for ResNet-50 and Vgg16, targeting latency as the primary optimization objective. Figure 12 shows latency, energy, and off-chip memory accesses for each approach across the two networks; each group of bars is normalized to the single-accelerator case ($n_{acc} = 1$) for that approach. Both FLIP2M and Tangram achieve substantial speedups as more accelerators are added; however, FLIP2M demonstrates slightly higher improvement factors (e.g., 9.75× vs. 8.96× for ResNet-50 and 16.83× vs. 14.06× for Vgg16 at $n_{acc} = 16$). This advantage primarily stems from FLIP2M's combined use of flexible per-layer resource allocation and inter-layer pipelining, which significantly reduces accelerator idle times and leverages better data locality at higher accelerator counts. For ResNet-50, FLIP2M exhibits worse energy scaling compared to Tangram due to residual blocks causing partial resource underutilization. Tangram's on-chip buffering optimization strategies can more effectively constrain resource usage within these blocks. Conversely, Vgg16 can better benefit from FLIP2M's flexible resource allocation for high accelerator

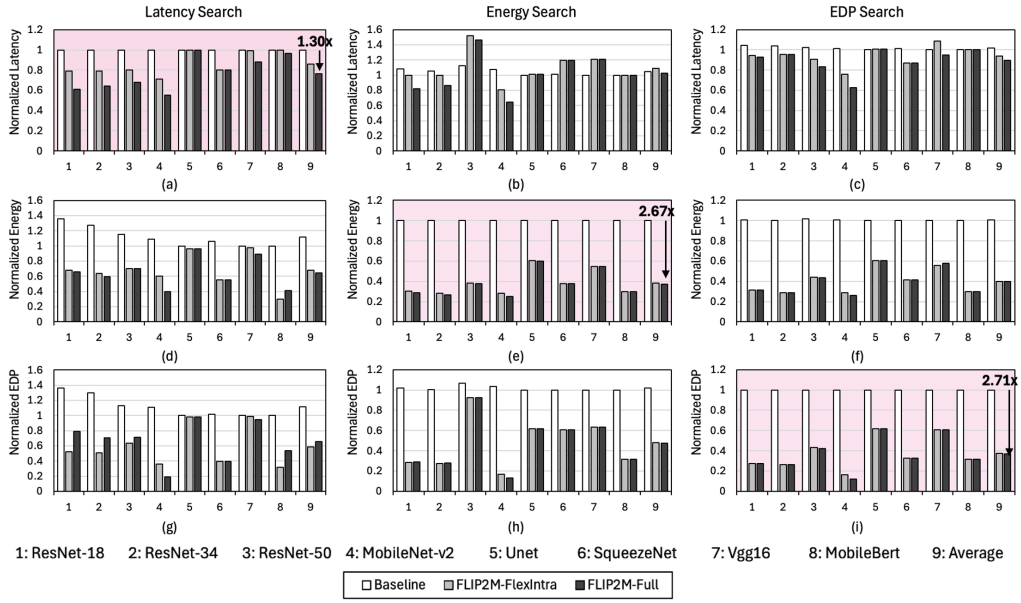

Fig. 11. End-to-end performance of single-model execution. Highlighted barplots indicate the main results (aligned optimization and evaluation metric).

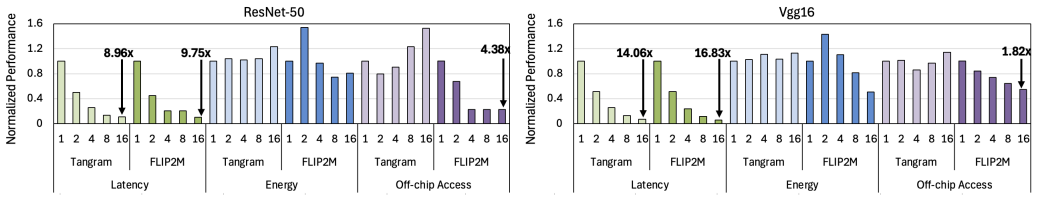

Fig. 12. Single-model scalability comparison of FLIP2M against tangram [15].

counts, owing to its homogeneous and stacked convolutional structure. FLIP2M provides substantial reductions in off-chip memory accesses at higher accelerator counts, achieving 4.38× and 1.88× reductions for ResNet-50 and Vgg16, respectively. A primary reason for this trend lies in FLIP2M's flexibility in distributing resources across layers within segments. Specifically, FLIP's configurable interconnect supports diverse P2P and multicast communication patterns, enabling more fine-grained resource allocation and on-chip data reuse. In contrast, Tangram primarily emphasizes buffering strategies that can help improve synchronization across accelerators assigned to different layers of the pipeline, but does not exploit interconnect flexibility to the same extent. As a result, Tangram misses opportunities for resource allocation optimizations that significantly reduce intermediate off-chip data transfers. Notably, its number of off-chip accesses holds relatively steady for Vgg16 and rises sharply for ResNet-50, as the number of accelerators increases.

6.3 Multi-Model Performance

We next evaluate FLIP2M on concurrent multi-model workloads. Figure 13 presents the performance of each primary metric under different optimization objectives for the AR/VR1, AR/VR2, and AR/VR3 workloads, using a fixed value of partition windows (E=10). Similar to the single-model experiments,

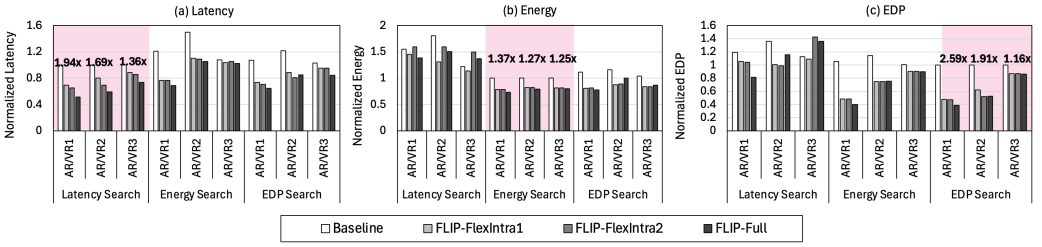

Fig. 13. End-to-end performance of multi-model execution.

we progressively enable FLIP2M's features under four configurations. Our *Baseline* disables inter-layer pipelining, and sets a maximum number of accelerator tiles and memory controllers each model can use based on its total number of MACs. *FLIP2M-FlexIntra1* removes the constraint on the maximum number of accelerator tiles that each model can use. *FLIP2M-FlexIntra2* further lifts the memory-controller limit, allowing each model flexible use of off-chip memory bandwidth. Finally, *FLIP2M-Full* adds inter-layer pipelining on top of *FLIP2M-FlexIntra2*. We again specify the optimization objective for the OASIS multi-model solver and report each metric normalized to its baseline.

When optimizing for latency, *FLIP2M-Full* yields clear gains over *Baseline* and the *FlexIntra* variants (Figure 13(a)). Running multiple models concurrently often inflates inference time due to contention for on-chip resources and off-chip bandwidth, yet *FLIP2M-Full* mitigates these penalties by efficiently pipelining layers and allocating accelerator tiles for each active segment. For AR/VR1 and AR/VR2, *FLIP2M-Full* achieves 1.94× and 1.69× speedups over the Baseline, respectively, significantly exceeding the average 1.30× latency improvement observed in the single-model experiments. While this improvement does come at higher energy cost (13.b), the overall EDP (13.c) remains competitive with the corresponding Baseline, reflecting how inter-layer pipelining effectively reduces idle accelerator time under multi-model contention.

Energy consumption across the various searches is shown in Figure 13(b) As in single-model deployments, the main driver of energy savings is intra-layer resource allocation; thus, *FLIP2M-FlexIntra* and *FLIP2M-Full* often achieve similar results. Under multi-model conditions, concurrency can further amplify memory bandwidth bottlenecks, which dampens overall energy reduction compared to the single model experiments. Nevertheless, compared to the Baseline approach, the FlexIntra configurations demonstrate the benefits of scaling resources to each layer's intensity. While the maximum savings are less pronounced than the 2.67× average from single-model energy searches, FLIP2M still shows meaningful reductions in energy of up to 1.37×. EDP results, depicted in Figure 13(c), follow similar trends, with more significant gains derived from the reduced execution time. The *FLIP2M-FlexIntra* variants typically improve substantially upon the *Baseline*, with *FLIP2M-Full* offering modest yet consistent additional benefits when off-chip memory accesses can be further reduced through inter-layer pipelining.

Although concurrency can limit certain gains, especially for memory-bound segments, the multi-model results align with the single-model trends. Flexible intra-layer parallelism substantially curtails idle compute, while inter-layer pipelining provides notable latency gains under resource contention. In fact, *FLIP2M-Full* exhibits higher latency improvements for multi-model workloads than single-model ones, confirming that dynamic resource and memory bandwidth partitioning and careful orchestration of segment depths become even more crucial when multiple models run concurrently. Conversely, energy scaling can be less dramatic, since concurrency often forces heavier resource usage or leaves certain memory-bound tasks unaffected by additional parallelism. Still, by tailoring accelerator tile and memory controller allocations layer-by-layer, the *FLIP2M-FlexIntra*

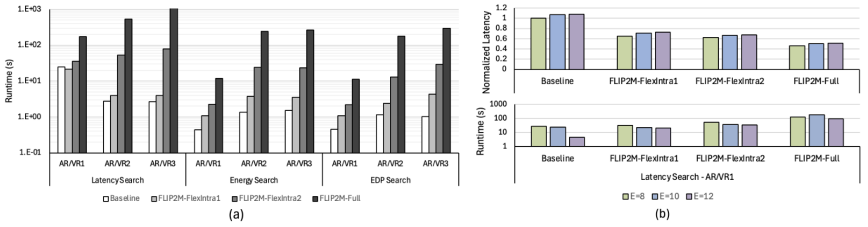

Fig. 14. (a) Multi-model solver execution time (b) Ablation study for # windows (E).

variants substantially outperform the Baseline in energy and EDP. Ultimately, these findings validate FLIP2M as a robust, holistic solution for multi-model AR/VR workloads, achieving meaningful gains across latency, energy, and EDP, even when facing heightened resource contention.

Figure 14(a) presents the runtime of the multi-model solver across the three objective searches and the four configurations. The execution time rises steadily as the search space expands from the *Baseline* to *FLIP2M-Full* for all three AR/VR workloads and for every optimization objective. This is because the *FLIP2M-Full* setting permits unconstrained accelerator tile and memory-controller allocation, as well as multi-layer segments. This forces the solver to examine a much larger set of execution modes at every decision point, which inevitably lengthens the search. Figure 14(b) reports the latency of AR/VR1 obtained with three different numbers of windows (E), along with the corresponding solver execution time. As expected, latency consistently improves when the number of windows is reduced, but at the expense of a longer solver execution time. Fewer windows aggregate more layers into each window, thereby expanding the search space and increasing the number of states the solver must explore. Other configurations and objective searches display a similar trend.

Comparison with SET. To validate FLIP2M's scalability in multi-model deployments, we compare it with SET [3], the current state-of-the-art framework for exploring intra-layer parallelism and inter-layer pipelining on tiled architectures. SET provides a rigorous mathematical formulation of the scheduling space and searches it with simulated annealing, but it cannot spatially partition resources: all accelerator tiles form a single pool, and layers from different models are executed by temporally multiplexing the entire fabric. FLIP2M, in contrast, can allocate disjoint groups of accelerator tiles and memory controllers to different segments and models, allowing several pipelines of layers to run concurrently. For this study, we extended the public SET release to support Vgg16 and ResNet-34 and constructed an AR/VR workload consisting of Vgg16, ResNet-50, and ResNet-34 (batch size = 4). For both frameworks, we set the EDP as the optimization objective and swept the number of accelerator tiles from 4 to 32 (SET cannot target $n_{acc} = 1,2$). Figure 15 reports the resulting EDP, energy, and latency. FLIP2M scales better in EDP at every tile count, reaching a 10.45× reduction at 32 tiles versus 9.35× for SET. The energy trends reveal the root cause: FLIP2M lowers energy as the fabric grows, by up to 1.69× at $n_{acc} = 32$. This is primarily because FLIP2M can apply inter-layer pipelining to multiple models simultaneously and keep resources utilized while avoiding redundant data movement. By contrast, SET processes only a single model at any given time; when heterogeneous layers are deployed, this leads to higher resource under-utilization. Consequently, its energy consumption rises slightly as additional tiles are added. SET displays better scalability for latency, largely because it allows segments deeper than three layers and employs an Eyeriss-style [7] tiled micro-architecture, that is, optimized for throughput. FLIP2M limits segment depth to three, both to bound the solver complexity and because, as mentioned in Section 3.2.2, deeper pipelines are demonstrated to yield diminishing returns. In addition, the design of the accelerator tile in our FLIP prototype is tuned primarily for energy efficiency rather than minimum latency. Nevertheless, the latency advantage of SET is offset by its higher energy consumption, so FLIP2M

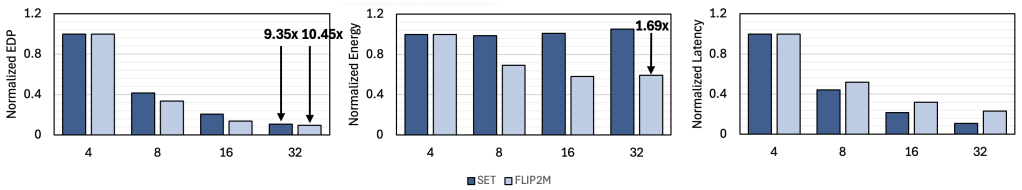

Fig. 15. Multi-model scalability comparison of FLIP2M against SET [3].

delivers superior EDP scalability, which is the critical metric for our edge-class multi-model AR/VR use case.

7 Related Work

Multi-Tenancy in Accelerators. The widespread adoption of DNNs has resulted in numerous works that address multi-model workloads in accelerator designs [2, 8, 16, 26, 28, 50]. Several works address multi-model execution by relying on *temporal partitioning*, where the full set of resources of the accelerator is allocated to one model at a time, while advanced scheduling and pre-emption techniques are utilized to balance fairness, throughput, and service-level agreements (SLAs). PREMA [10] introduces fine-grained pre-emptive scheduling, in which dynamic task-score adjustment maintains both fairness and SLA compliance. AI-MT [1] proposes a load-balancing policy that reduces resource under-utilization through a combination of memory-block prefetching and compute-block merging. Sparse-DySta [13] couples a static scheduler—using sparsity-aware latency estimates to derive initial priorities—with a dynamic runtime component that updates those priorities based on observed behaviour. Layerweaver [36] further improves utilization and throughput by deploying a greedy scheduler that heuristically estimates the cost of alternative scheduling decisions.

A second line of work augments temporal partitioning with *spatial partitioning*, dividing the on-chip fabric in order to run multiple models concurrently. Planaria [16] and Dataflow-Mirroring [9] propose different variants of dynamic architecture fission: an omni-directional systolic array paired with a reconfigurable interconnect can be split into independent sub-arrays on demand, each assigned to a different model or task. MOCA [26] optimizes memory bandwidth partitioning to reduce contention. Additionally, heterogeneous accelerator designs such as HDA [28] and JNPU [50] incorporate various dataflow strategies for enhanced data reuse. Li et al. [30] propose a tensor-arrangement framework that tackles data-dimension conversion from the linear off-chip address space to the parallel on-chip tensor execution. Their approach decouples data memory from compute through a block-based, adaptive data layout that dynamically matches data dimensions during processing, thereby accommodating dataflow variability. MAGMA [21] and H3M [53] propose custom genetic algorithm-based optimization methods for the multi-model mapping problem. M2M [54] proposes a communication-aware block mapping tool for chiplet-based accelerators using a simulated annealing algorithm. However, these approaches have not considered inter-layer pipelining.

Inter-Layer Pipelining. Several works have explored inter-layer pipelining as an optimization technique [15, 17, 44]. DeFiNES [33] proposes a unified framework which extends the scheduling space of existing analytical models [27, 31, 37, 49] with deep layer-fused execution, where the feature maps are split into serialized computation nodes, to allow for intermediate on-chip feature forwarding across different layers. STREAM [44] extends this framework to multi-accelerator platforms, while DeepFrack [17] primarily emphasizes efficient on-chip caching for improved data reuse. Colleman et al. [11] extends STREAM with support for transpose and softmax layers, therefore enabling layer-fusion for transformers. Tangram [15] proposes several optimization strategies to reduce buffering requirements for inter-layer pipelining. SET [3] provides a rigorous mathematical notation for describing the

scheduling space created by inter-layer pipelining on tiled architectures and introduces a simulated-annealing–based framework to explore that space. Gemini [4] extends the SET framework to large-scale chiplet accelerators by augmenting the mapping engine to minimize expensive **die-to-die** (**D2D**) traffic, and by incorporating a monetary-cost evaluator that estimates the fabrication expense of different architectural candidates. Nevertheless, these methods lack flexibility to combine intra-layer parallelism and inter-layer pipelining and do not support spatial partitioning for multi-model execution.

Combined Multi-Model and Inter-Layer Pipelining. Limited research has integrated multi-model execution with inter-layer pipelining. SCAR [35] addresses multi-model execution in chiplet-based architectures, pipelining segments across different chiplets, but lacks flexibility in tailoring intra-layer resource allocation while pipelining layers. In contrast, our proposed FLIP2M uniquely combines intra-layer parallelism and inter-layer pipelining with robust multi-model support, significantly enhancing flexibility and efficiency compared to previous approaches.

Multi-Tenancy in GPUs Contemporary GPUs support spatial resource partitioning via Multi-Instance GPU (MIG) [34], which divides a single device into as many as seven fully isolated instances, each with its own set of **streaming multiprocessors** (**SMs**), L2 cache banks, and HBM memory slices—conceptually similar to FLIP's "accelerator-tile + memory-controller" groups. MIG, however, is available only on high-end A100/H100 data-center parts whose power and area envelopes are roughly two orders of magnitude larger than those acceptable for edge-class AR/VR platforms. In contrast, Jetson-class GPUs (Nano, Xavier NX, and Orin NX) offer no hardware support for partitioning the on-chip interconnect or DRAM interface into instance-private lanes; all the compute units contend for the same DRAM channels and share a global warp scheduler. Consequently, prior edge-GPU work addresses multi-model execution almost exclusively through temporal multiplexing [18, 47].

Workload Dynamicity DREAM [25] demonstrates that dynamic workloads make system load unpredictable, challenging ML accelerators that depend on deterministic latency estimates for static schedules. To cope with variability at the task, model, and operator levels, DREAM continuously monitors the environment request stream and system state, scores candidate layer–accelerator mappings with pre-profiled latency/energy data, adaptively tunes its scheduling parameters, and dynamically dispatches or drops tasks to keep all models within their real-time deadlines. OASIS does not support dynamic inter-model dependencies, but its main contribution is orthogonal to these run-time challenges. We focus on an accelerator architecture whose regular 2-D mesh enables a flexible combination of intra-layer parallelism and inter-layer pipelining, not addressed by DREAM, and on formulating mapping and scheduling as a multi-mode RCPSP solved for latency-optimal, energy-optimal, or EDP-optimal static deployments. The resulting FLIP2M infrastructure can serve as a foundation on which future work can build the kind of run-time monitoring and adaptive re-scheduling pioneered by DREAM—re-using our cost model and optimization framework while extending them to fully dynamic **real-time multi-model** (**RTMM**) settings.

8 Conclusion

Combining intra-layer parallelism and inter-layer pipelining is critical for deploying multi-model AR/VR workloads on tiled accelerator architectures; in this article, we developed FLIP2M to address the challenges associated with doing so. Intra-layer parallelism offers the flexibility to adapt to the heterogeneous demands from different layers, while inter-layer pipelining alleviates the burden of costly off-chip accesses. The FLIP accelerator fabric supports these optimizations through its flexibility for both coarse-grained parallelism and on-chip communication patterns; its high-level architectural approach can support different accelerator implementations. We also presented OASIS, an optimization framework to help navigate the enormous mapping space when deploying both single-model and multi-model workloads on the FLIP architecture. We prototyped an instance of FLIP, implemented as a 49-tile SoC architecture, on an FPGA device using the ESP platform.

Our experimental results show that, when running real multi-model AR/VR workloads, FLIP2M achieves up to a 1.94× improvement in latency, a 1.37× reduction in energy consumption, and a 2.59× improvement in the EDP compared to a FLIP baseline configuration. We plan to publicly release FLIP2M, including our implementation of the FLIP prototype and the OASIS optimization framework, as open-source artifacts to support future research on this topic.

References

[1] E. Baek, D. Kwon, and J. Kim. 2020. A multi-neural network acceleration architecture. In *Proceedings of theACM/IEEE 47th Annual Intl. Symp. on Computer Architecture (ISCA)*. 940–953.

[2] F.G. Blanco, E. Russo, M. Palesi, D. Patti, G. Ascia, and V. Catania. 2024. A deep reinforcement learning based online scheduling policy for deep neural network multi-tenant multi-accelerator systems. In *Proceedings of the Design Automation Conf. (DAC)*. 1–6.

[3] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma. 2023. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the Intl. Symp. on Computer Architecture (ISCA)*. Article 13, 17 pages.

[4] J. Cai, Z. Wu, S. Peng, Y. Wei, Z. Tan, G. Shi, M. Gao, and K. Ma. 2024. Gemini: Mapping and architecture co-exploration for large-scale DNN chiplet accelerators. In *Proceedings of the 2024 IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 156–171.

[5] L. P. Carloni. 2015. From latency-insensitive design to communication-based system-level design. *Proc. of the IEEE* 103, 11 (2015), 2133–2151.

[6] L. P. Carloni. 2016. The case for embedded scalable platforms. In *Proceedings of the Design Automation Conf. (DAC)*. 17:1–17:6.

[7] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd Intl. Symp. on Computer Architecture (ISCA)*. 367–379.

[8] K.-L Chiu, G. Eichler, C.-T. Lin, G.-D. Guglielmo, and L.-P. Carloni. 2024. WOLT: Transparent deployment of ML workloads on lightweight many-accelerator architectures. In *Proceedings of the Intl. Conf. on Computer Design (ICCD)*. 637–644.

[9] J. Choi, Y. Ha, J. Lee, S. Lee, J. Lee, H. Jang, and Y. Kim. 2023. Enabling fine-grained spatial multitasking on systolic-array NPUs using dataflow mirroring. *IEEE Trans. on Computers* 72, 12 (2023), 3383–3398.

[10] Yujeong Choi and Minsoo Rhu. 2020. PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 220–233. DOI: https://doi.org/10.1109/HPCA47549.2020.00027

[11] Steven Colleman and others. 2024. Optimizing layer-fused scheduling of transformer networks on multi-accelerator platforms. In *2024 25th Intl. Symp. on Quality Electronic Design (ISQED)*. 1–6. DOI: https://doi.org/10.1109/ISQED60706.2024.10528689

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

[13] H. Fan, S.-I. Venieris, A. Kouris, and N.-D. Lane. 2023. Sparse-DySta: Sparsity-aware dynamic and static scheduling for sparse multi-DNN workloads. In *Proceedings of the 56th Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*. 353–366.

[14] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 751–764.

[15] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 807–820.

[16] S. Ghodrati, S. Ghodratiand, B.-H. Ahn, J.-K. Kim, S. Kinzer, B.-R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N.-S. Kim, C. Young, and H. Esmaeilzadeh. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *Proceedings of the IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*. 681–697.

[17] T. Glint, M. Pechimuthu, and J. Mekie. 2024. DeepFrack: A comprehensive framework for layer fusion, face tiling, and efficient mapping in DNN hardware accelerators. In *Proceedings of the Design, Automation, and Test in Europe Conf. (DATE)*. 1–6.

[18] L. Han, Z. Zhou, and Z. Li. 2024. Pantheon: Preemptible multi-DNN inference on mobile edge GPUs. In *Proceedings of the 22nd Annual Intl. Conf. on Mobile Systems, Applications and Services (MobiSys)*. 465–478.

[19] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[20] F. N. Iandola and others. 2016. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <1MB model size. *ArXiv* abs/1602.07360, (2016). Retrieved from https://api.semanticscholar.org/CorpusID:14136028
[21] S. Kao and T. Krishna. 2022. MAGMA: An optimization framework for mapping multiple DNNs on multiple accelerator cores. In *Proceedings of the IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 814–830.
[22] S. Karl, A. Symons, N. Fasfous, and M. Verhelst. 2023. Genetic algorithm-based framework for layer-fused scheduling of multiple DNNs on multi-core systems. In *Proceedings of the Design, Automation & Test in Europe Conf. (DATE)*. 1–6.
[23] B. Khailany, E. Krimer, R. Venkatesan, J. Clemons, J.-S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y.-S. Shao, S. Srinath, C. Torng, S.-L. Xi, Y. Zhang, and B. Zimmer. 2018. INVITED: A modular digital VLSI flow for high-productivity soc design. In *Proceedings of the Design Automation Conf. (DAC)*. 1–6.
[24] S. Kim, J. Zhao, K. Asanovic, B. Nikolic, and Y.-S. Shao. 2023. AuRORA: Virtualized accelerator orchestration for multi-tenant workloads. In *Proceedings of the Intl. Symp. on Microarchitecture (MICRO)*. 62–76.
[25] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra. 2023. DREAM: A dynamic scheduler for dynamic real-time multi-model ML workloads. In *Proceedings of the Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 73–86.
[26] S. Kim, H. Genc, V.-V. Nikiforov, K. Asanovic, B. Nikolic, and Y.-S. Shao. 2023. MoCA: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *Proceedings of the Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 828–841.
[27] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar. 2020. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro* 40, 3 (2020), 20–29.
[28] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra. 2021. Heterogeneous dataflow accelerators for Multi-DNN workloads. In *Proceedings of the Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 71–83.
[29] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda, C. Banbury, M. Mazumder, L. Lai, A. Sirasao, T. Krishna, H. Khaitan, V. Chandra, and V.-J. Reddi. 2023. XRBench: An extended reality (XR) machine learning benchmark suite for the metaverse. *Proc. of Machine Learning and Systems* 5 (2023), 1–20.
[30] C. Li, X. Fan, X. Wu, Z. Yang, M. Wang, M. Zhang, and S. Zhang. 2022. Memory-computing decoupling: A DNN multitasking accelerator with adaptive data arrangement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4112–4123.
[31] Linyan M., Pouya H., Vikram J., Sebastian G., and Marian V.. 2020. ZigZag: A memory-centric rapid DNN accelerator design space exploration framework. arxiv:2007.11360. Retrieved from https://arxiv.org/abs/2007.11360 (2020).
[32] P. Mantovani, D. Giri, G.-D. Guglielmo, L. Piccolboni, J. Zuckerman, E.-G. Cota, M. Petracca, C. Pilato, and L.-P. Carloni. 2020. Agile SoC development with open ESP. In *Proceedings of the Intl. Conf. on Computer-Aided Design*.
[33] L. Mei, K. Goetschalckx, A. Symons, and M. Verhelst. 2023. DeFiNES: Enabling fast exploration of the depth-first scheduling space for DNN accelerators through analytical modeling . In *Proceedings of the 2023 IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 570–583.
[34] NVIDIA. 2022. Multi-Instance GPU User Guide. Retrieved from https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html. (2022).
[35] M. Odema, L. Chen, H. Kwon, and M.-A.-A. Faruque. 2024. SCAR: Scheduling multi-model AI workloads on heterogeneous multi-chiplet module accelerators. In *Proceedings of the Intl. Symp. on Microarchitecture (MICRO)*. 565–579.
[36] Y. Oh, S. Kim, Y. Jin, S. Son, J. Bae, J. Lee, Y. Park, D.-U. Kim, T.-J. Ham, and J.-W. Lee. 2021. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *Proceedings of the 2021 IEEE Intl. Symp. on High-Performance Computer Architecture (HPCA)*. 584–597.
[37] A. Parashar, P. Raina, Y.-S. Shao, Y.-H. Chen, V.-A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S.-W. Keckler, and J. Emer. 2019. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the 2019 IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*. 304–315.
[38] O. Ronneberger, P. Fischer, and T. Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the Intl. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 234–241.
[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 4510–4520.
[40] K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd Intl. Conf. on Learning Representations ICLR 2015*. http://arxiv.org/abs/1409.1556
[41] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen. 2019. Hypar: Towards hybrid parallelism for deep learning accelerator array. In *Proceedings of the Intl. Symp. on High Performance Computer Architecture (HPCA)*. 56–68.
[42] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen. 2020. Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In *Proceedings of the Intl. Symp. on High Performance Computer Architecture (HPCA)*. 342–355.
[43] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. 2020. MobileBERT: A compact task-agnostic BERT for resource-limited devices. *arXiv:2004.02984*. Retrieved from https://arxiv.org/abs/2004.02984 (2020).

[44] A. Symons and others. 2025. Stream: Design space exploration of layer-fused DNNs on heterogeneous dataflow accelerators. *IEEE Trans. on Computers* 74, 1 (2025), 237–249.

[45] F. Brian Talbot. 1982. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science* 28, 10 (1982), 1197–1210.

[46] R. Venkatesan, Y.-S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, Y. Zhang, B. Zimmer, W.-J. Dally, J. Emer, S.-W. Keckler, and B. Khailany. 2019. MAGNet: A modular accelerator generator for neural networks. In *Proceedings of the Intl. Conf. on Computer-Aided Design (ICCAD)*. 1–8.

[47] K. Wang, J. Cao, Z. Zhou, and Z. Li. 2024. SwapNet: Efficient swapping for DNN inference on edge AI devices beyond the memory budget. *IEEE Trans. on Mobile Computing* 23, 9 (2024), 8935–8950.

[48] C.J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, and M. Dukhan. 2019. Machine learning at facebook: Understanding inference at the edge. In *Proceedings of the Intl. Symp. on High Performance Computer Architecture (HPCA)*. 331–344.

[49] Y. Wu, J. S. Emer, and V. Sze. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *Proceedings of the 2019 IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*. 1–8.

[50] J. Yang, S. Lim, S. Lee, J.-Y. Kim, and J.-Y. Kim. 2023. JNPU: A 1.04 TFLOPS Joint-DNN training processor with speculative cyclic quantization and triple heterogeneity on microarchitecture/precision/dataflow. In *Proceedings of the European Solid State Circuits Conf. (ESSERC)*. 349–352.

[51] Y.J. Yoon, N. Concer, M. Petracca, and L.P. Carloni. 2013. Virtual channels and multiple physical networks: Two alternatives to improve NoC performance. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 32, 12 (2013), 1906–1919.

[52] F. Zaruba and L. Benini. 2019. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-Bit RISC-V core in 22-nm FDSOI technology. *IEEE Trans. on Very Large Scale Integration Systems* 27, 11 (2019), 2629–2640.

[53] S. Zeng, G. Dai, N. Zhang, X. Yang, H. Zhang, Z. Zhu, H. Yang, and Y. Wang. 2023. Serving multi-DNN workloads on FPGAs: A coordinated architecture, scheduling, and mapping perspective. *IEEE Trans. on Computers* 72, 5 (2023), 1314–1328.

[54] J. Zhang, X. Wang, Y. Ye, D. Lyu, G. Xiong, N. Xu, Y. Lian, and G. He. 2024. M2M: A fine-grained mapping framework to accelerate multiple DNNs on a multi-chiplet architecture. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 32, 10 (2024), 1864–1877.

[55] X. Zhang, C. Hao, P. Zhou, A. Jones, and J. Hu. 2022. H2H: Heterogeneous model to heterogeneous system mapping with computation and communication awareness. In *Proceedings of the Design Automation Conf. (DAC)*. 601–606.

Received 12 August 2025; revised 12 August 2025; accepted 12 August 2025