SCAR: Scheduling Multi-Model AI Workloads on Heterogeneous Multi-Chiplet Module Accelerators

Mohanad Odema
Univ. of California, Irvine
Irvine, CA, USA
modema@uci.edu

Luke Chen
Univ. of California, Irvine
Irvine, CA, USA
panwangc@uci.edu

Hyoukjun Kwon
Univ. of California, Irvine
Irvine, CA, USA
hyoukjun.kwon@uci.edu

Mohammad Abdullah Al Faruque
Univ. of California, Irvine
Irvine, CA, USA
alfaruqu@uci.edu

2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 979-8-3503-5057-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MICRO61859.2024.00049

***Abstract*—Emerging multi-model workloads with heavy models like recent large language models significantly increased the compute and memory demands on hardware. To address such increasing demands, designing a scalable hardware architecture became a key problem. Among recent solutions, the 2.5D silicon interposer multi-chip module (MCM)-based AI accelerator has been actively explored as a promising scalable solution due to their significant benefits in the low engineering cost and composability. However, previous MCM accelerators are based on homogeneous architectures with fixed dataflow, which encounter major challenges from highly heterogeneous multi-model workloads due to their limited workload adaptivity.**

Therefore, in this work, we explore the opportunity in the heterogeneous dataflow MCM AI accelerators. We identify the scheduling of multi-model workload on heterogeneous dataflow MCM AI accelerator is an important and challenging problem due to its significance and scale, which reaches O(10^{56}) even for a two-model workload on 6x6 chiplets. We develop a set of heuristics to navigate the huge scheduling space and codify them into a scheduler, SCAR, with advanced techniques such as inter-chiplet pipelining. Our evaluation on ten multi-model workload scenarios for datacenter multitenancy and AR/VR use-cases has shown the efficacy of our approach, achieving on average 27.6% and 29.6% less energy-delay product (EDP) for the respective applications settings compared to homogeneous baselines.

***Index Terms*—AI accelerators, Multichip modules, Chiplets, Scheduling algorithms, Performance analysis.**

I. Introduction

Recent artificial intelligence (AI) inference workloads have increased their scale in both of the model size (e.g., large language models [7], [69]) and the number of models deployed together (e.g., augmented and virtual reality; AR/VR [38]), which constructs multi-model workloads with heavier models than those in the past. Such trends led to heavy demands on compute capabilities in AI hardware from edge to cloud devices. As an approach to scale up the hardware for AI and increase the compute capability, chiplet-based multi-chip module (MCM) package has emerged as a promising solution [55], [64], [68], [71]. Such MCM packages facilitate the scaling of AI hardware based on their composability and cost-effectiveness, unlike monolithic designs, which are often constrained by fabrication yields, power, heat, and other engineering costs such as verification [50].

Researchers have actively explored the MCM for AI, focusing on the dataflow mapping (i.e., loop ordering, parallelization, and tiling) of each layer and workload orchestration onto chiplets considering the network-on-package (NoP) and other communication constraints [55], [64], [68], [71]. For example, Simba [64] proposed a scalable MCM inference architecture that enables chiplets to either act as standalone inference engines or collaborate as groups for a layer. Although such works have successfully delivered promising performance and energy efficiency than monolithic designs, they mostly focused on *single-model* workloads targeting *homogeneous* chiplets. Unlike single-model workloads, multi-model workloads introduce major challenges to such homogeneous MCMs because of the ML operator heterogeneity (e.g., operator types and tensor sizes) and resulting diverse dataflow preferences [37]. Also, multi-model workloads often involve model level dependency and concurrency [34], [37], [38], [51], [56], which adds complex considerations to the scheduling problem.

Therefore, considering the new trend with multi-model AI workloads in industry, such as multi-tenancy [23], [40], [72] and AR/VR [38], we explore heterogeneous chiplet-based MCM with AI accelerator chiplets with various dataflows, as a future-proof option. To exploit the benefits of heterogeneous MCM accelerators, we consider inter-layer pipelining to enhance in-package data reuse and reduce offchip traffic. We formulate the scheduling problem and develop effective heuristics to navigate the huge scheduling space, whose problem scale is as big as O(10^{56}) even for a two-model workload (ResNet-50 [24] and UNet [63]) on a 6x6 chiplet MCM AI accelerator system (as in Simba [64]).

We evaluate ten MCMs including seven heterogeneous MCMs on ten multi-model scenarios: the first five scenarios are curated using MLPerf inference benchmark [62] representing datacenter multi-tenancy scenarios. The models are selected based on recent datacenter model usage trends [23], [29] and the trend of language model adoptions (e.g., GPT-L [60]), future-proofing emerging AI workloads such as AI assistant [47]. The other five scenarios are curated for AR/VR usage scenarios from XRBench as a practical use case for edge multi-model workloads [38].

The evaluation results show that heterogeneous MCM combined with our scheduling method is promising for heavy multi-model workloads, which is projected by recent trend. Compared to the homogeneous MCM [64] running NVDLA [52] and Shi-diannao [16] style dataflows, heterogeneous MCM, on average, achieved 27.6% and 29.6% less energy-delay product (EDP) in each domain, respectively. We also showcase that our scheduler can identify schedules that can reduce EDP to 0.3× that of single-model schedulers like NN-baton [68]. We summarize our contributions as follows:

979-8-3503-5057-9/24/$31.00 ©2024 IEEE
DOI 10.1109/MICRO61859.2024.00049

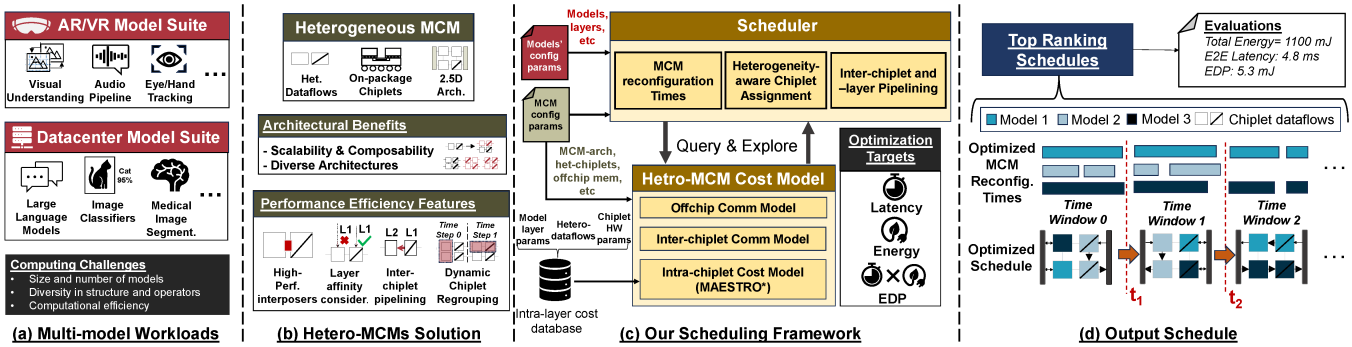

Fig. 1. An overview of this work: (a) Emerging Multi-model workloads have introduced new challenges for AI hardware. (b) Heterogeneous MCMs present a promising solution to scale with multi-model workloads with some considerations. (c) Our proposed scheduling framework addresses said challenges to explore the heterogeneous scheduling space. (d) Provided Solutions provide optimized spatio-temporal scheduling strategies for the multi-model workloads.

- As a promising future-proof architecture for heavy multi-model workloads, we explore heterogeneous dataflow MCM for emerging AI workloads with multiple models running concurrently.
- We formulate the MCM AI accelerator scheduling problem into a multi-tiered optimization problem to address intractably large scheduling space.
- Based on the formulation, we develop a scheduler that thoroughly considers heterogeneous MCM and multi-model workloads. The scheduler employs advanced scheduling techniques (inter-layer pipelining, dynamic chiplet regrouping) with resource allocation tree representation [8].
- We codify our scheduling method and integrate it with a heterogeneous MCM AI accelerator cost model. We extend MAESTRO [35], [36] to model the latency and energy of MCM accelerators.
- We analyze the costs and benefits of heterogeneous dataflow MCM using industry use case-inspired multi-model workloads and present the importance of the scheduling problem.

II. Background and Motivation

A. Multi-model AI Workloads

Multi-model AI Workloads. The success of AI algorithms in individual tasks (e.g., hand tracking, depth estimation, speech recognition) led to the emergence of multi-model AI workloads, which include multi-tenant workloads at data centers [23], [40], [72] and real-time multi-model workloads such as AR/VR [38]. We summarize example multi-model AI workloads from industrial use cases in Table III. The models in such workloads are diverse in terms of the tasks and input modalities. For example, an industrial data center multi-tenant AI workload suite [23] includes a face recognition model based on support vector machine, recommendation models based on multi-layer perceptron, and a speech recognition model based on recurrent neural network (RNN). More recent workloads in data center AI workload include large language models [48], which adds more heterogeneity to the multi-model AI workloads. As discussed in prior works [37], [38], such multi-model workloads involve high heterogeneity in AI operators (or layers), which is one of the major challenges to accelerators that specialize the architecture and dataflow for a specific set of workloads.

B. Scheduling AI workloads on AI Hardware and MCMs

Scheduling AI workloads considers the assignment of computations (e.g., model, layer, or tile) to target hardware platforms and their constituent computing units. We provide a brief on the state of AI workloads scheduling practices.

Scheduling on CPU/GPU systems. modern systems (servers) typically employ GPUs and/or CPUs for inference services [4], [12], [13], [22], [23], [54], [65], [72], [76]. Most of these computing units are based on homogeneous cores – or simple heterogeneity such as big and little cores in CPUs, or CUDA and Tensor cores in GPUs. Traditionally, the scheduling in such settings is concerned with the coarse assignment of models to computing units, leaving the operator assignments to be performed in a direct manner (e.g., all GEMM operations to Tensor Cores in a GPU). As multi-model workloads proliferated, new features (e.g., GPU sharing) emerged to improve inference services for small-batch inference tasks [11], [12]. Still, the limited programmer/compiler control and the cache-based memory systems restrict CPUs/GPUs from engaging multi-model workload scheduling on a finer granularity.

Scheduling on customized AI accelerators. Customized AI accelerators (Google's TPU [29] Meta's MTIA [18]) enable full programmer/compiler control over memory operations (when and what to read/write, when and what to evict, etc). AI accelerators typically employ scratchpad memory-based systems to support deterministic low-level activities [21], [36], [37], [57]. AI accelerators are also integrated into edge hardware (e.g., NPU in Apple Vision Pro's M2 chip [2]).

Scheduling on MCM AI Accelerators. To scale with the rising compute demands of modern AI workloads, multi-chip modules (MCMs) have emerged as viable approach enabling the integration of composable, small functional dies (chiplets) on the package level to build a larger system, where they are connected together via on-package links typically through silicon interposer or organic substrates to create a network-on-package (NoP) [5], [30], [70]. Through enabling scalability via adjusting the number of chiplets on the package, as well as low verification costs [50], many chiplet-based systems

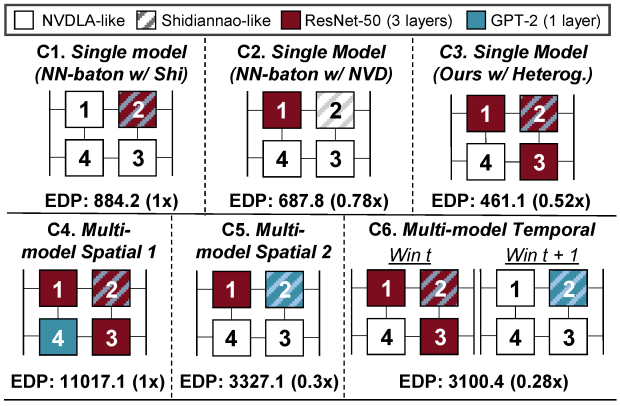

Fig. 2. Motivational study on a 2×2 MCM AI accelerator using batch size of 1 for 3 layers from the second ResNet-50 block and the first feed forward layer from GPT-2. Each chiplet has 4096 PEs and 10 MB L2 shared memory (Full from Section V). NN-baton [68] considers partitioning computation across chiplets only when not enough resources exist.

have been developed for scalable deep learning inference: Simba [64] is one comprising 36 chiplets, each containing 16 processing engines to deliver up to 128 TOPs computing capability. Another example is Tesla's DOJO chiplet-based architecture capable of scaling to exaFLOP supercomputers for large-scale machine learning. The scaling in chiplet sizes, architectures, and computational capabilities has enabled support for serving multi-model workloads together on the same MCM system with a finer degree of scheduling granularity (operator, tiles). However, multi-model schedulers face new challenges compared to their single-model counterparts considering the increased memory footprints, bandwidth contention, etc.

C. Motivational Example

We consider the NN-baton [68] as our baseline scheduler as it targets scheduling single model workloads on multi-chiplet accelerators. NN-baton proposes to partition a single model workload across several chiplets whenever its computational demands exceeds a single chiplet's capacity, and employs a unified dataflow across the chiplets. As heterogeneous accelerators proliferate [37], [77], chiplets technology has facilitated their integration on the package level. Consider a small heterogeneous 2×2 MCM containing 3 NVDLA-like (weight stationary) and 1 Shidiannao-like (output stationary accelerators, and consider a small multi-model workload constituting 3 layers from the second ResNet-50 block and one fully connected layer from GPT-L. We analyze the schedules yielded through NN-baton and our scheduler as follows.

Single model case. We show the single model scheduling results for the ResNet-50 workload in Figure 2 [A1-A3]. As each chiplet possess sufficient resources to process the ResNet-50 workload, NN-baton schedules the workload onto a single chiplet. As shown, scheduling the ResNet-50 workload to the NVDLA-like chiplet (A2) experiences 0.78× the EDP as that from the Shidiannao-like chiplet (A1). However, a more nuanced schedule (A3) identified through our scheduler leverages heterogeneity by distributing the ResNet-50 layers across the heterogeneous chiplets, sustaining 0.52× less EDP than (A1) through catering to individual layer affinities.

Multi-model case., In Figure 2 (B1-B3), NN-baton (B1) is agnostic to the heterogeneous MCM composition, executing each model workload sequentially on its starting chiplet 1. We show two schedules that are sampled through our schedules: (i) In (B2), our scheduler can spatially distribute the ResNet-50 and GPT-2 workloads across the NVDLA-like and Shidiannao-like accelerators, respectively, leading the EDP to be 0.3× that in (B1). (ii) In (B3), Our scheduler recognizes a spatio-temporal optimization to leverage the heterogeneous chiplet pipelining for the ResNet-50 in one time window, and then schedule the GPT-2 layer on the Shidiannao layer in the following time window, leading EDP to be 0.28× that of (B1).

D. Complexity of the multi-model scheduling space

To understand the scale of the multi-model scheduling problem, we analyze its search space complexity. Let a multi-model workload constitute N models, each model containing L_i layers, and $L{=}\sum_i^N L_i$. Let C be the total number of accelerator chiplets on an MCM. Then, a characterization of the multi-model scheduling space can be given as $\mathcal{O}(C^L \times \frac{L}{L_1!L_2!\cdots L_N!})$.

The first term (C^L) covers the set of possible chiplet assignments for each layer (spatial complexity); whereas the multinomial coefficient ($\frac{L}{L_1!L_2!\cdots L_N!}$) covers the number of ways to interleave multiple sequences of layers, while maintaining the layer dependencies for each model. In the small motivational example above, this complexity accounts for a total of $\mathcal{O}(1536)$ scheduling possibilities. If we consider a more practical case involving a ResNet-50 and UNet models ($L1 = 50$ and $L2 = 23$) on a full Simba system ($C = 36$), the complexity becomes $\sim \mathcal{O}(10^{56})$, showcasing an exponential rise as the models grow in number and complexity.

E. Summary on the challenges of multi-model scheduling

We summarize the unique scheduling challenges for multi-model workloads compared to their single-model counterparts:

- **Layer sequence permutations.** In a single model scenario, the space of computation assignments is defined based on dependent layer sequences from a single model. Whereas in a multi-model scenario, independent layer sequences from different models also exist, compounding the decision space complexity as a result of permutations.
- **Spatial mapping conditioning.** The quality of one model's schedule is affected by the spatial mappings of other models' mappings due to resource availability, bandwidth contention, added data travel times, and so on.
- **Heterogeneous Integration Trade-offs.** Performance efficiency depends on the underlying pattern of heterogeneous integration, and the diversity within and across model workloads imply that no single pattern fits all.

To address the challenges and search complexity, one approach is to formulate the problem as a multi-level decision problem where each decision subspace is a tractable problem [10], [37]. We adopt a similar approach and formulate the MCM multi-model workload scheduling as multiple-level decision problem, as shown in Figure 3. We detail our problem formulation and performance modeling methodology next.

TABLE I
NOTATION USED IN THE FORMULATION.

Notation	Description
Sc	Multi-model workload scenario
m_i	the i-th model from scenario Sc
$layer_{i,j}$	the j-th layer from model i
H	MCM hardware
C	Set of accelerator chiplets on H
c_i	the i-th chiplet from C
DF	set of supported dataflows on H
n_{df_i}	Number of chiplets adopting the i-th dataflow
$BW_{offchip}$	offchip bandwidth
BW_{nop}	Network-on-package bandwidth
df	Dataflow
N_{PE}	Number of processing engines
BW_{noc}	Network-on-chip bandwidth
Sz_{mem}	The memory size in c
$TW(Sc)$	The set of time windows for Sc
$tw(Sc)$	an execution time window for Sc on H
T_s	time window start
T_{tw}	time window end
$L(tw(Sc))$	set of layers executable on H during $tw(Sc)$
SG	Set of all valid segments for $tw(Sc)$
$sg(tw(Sc))$	a layer segment from $L(tw(Sc))$
$Sp_{tw}(Sc)$	Time window partioning space for Sc
$Sp_{sg}(tw)$	Layer segmentation space at tw
$SS_{TW}(tw(Sc), H)$	The scheduling space for $tw(Sc)$ on H
$SS_{Sc}(H)$	The overall scheduling space for Sc on H
$sched(Sc, H)$	A scheduling instance for Sc on H
$Lat^i_j(\mathbb{A})$	Latency evaluation for $\mathbb{A}$ given identifiers i, j
$E^i_j(\mathbb{A})$	Energy evaluation for $\mathbb{A}$ given identifiers i, j
Sz_{data}	Size of transmission data
n_{hops}	Number of hops from src to destination
n_{splits}	Number of time window splits

III. SYSTEM MODELING AND PROBLEM FORMULATION

To develop a systematic approach to navigate complex search space, we formulate the scheduling problem of multi-model workloads on a heterogeneous MCM AI accelerator.

A. Base Formulation

To formulate the MCM scehduling problem, we first define multi-model workload scenario (Sc) and MCM hardware (H).

We formulate the workload in the granularity of layers in each model. Therefore, we formulate a multi-model workload scenario (Sc) as the collection of layers in the models included in the scenario. Letting the number of models included in Sc as $|Sc|$ and the number of layers included in a model m as $|m|$, we define Sc as follows:

Definition 1. ***Multi-model Workload Scenario (Sc)***

$$Sc = \{layer_{i,j} | 0 < i \leq |Sc|, 0 < j \leq |m_i|\}$$

where $layer_{(i,j)}$ refers to the j-th layer of model i in Sc.

AI accelerator chiplets consist of a PE array, memory, and on-chip interconnection among memory and PEs. In addition to them, we also include the dataflow in the formulation to model heterogeneous chiplet MCM AI accelerator. Accordingly, we define an AI accelerator chiplet (c) as follows:

Definition 2. ***AI Accelerator Chiplet (c)***

$$c = \{df, N_{PE}, BW_{noc}, BW_{mem}, Sz_{mem}\}$$

In Definition 2, df refers to the dataflow, N_{PE} is the number of PEs, BW_{noc} is the NoC bandwidth, BW_{mem} is the chiplet-level shared memory bandwidth, and Sz_{mem} is the memory size in c.

Based on the definition of the chiplet, we formulate the MCM accelerator as the set of chiplets ($C = \{c_1, c_2, ..., c_{N_{cpl}}\}$), NoP, and off-chip interface as follows:

Definition 3. ***MCM AI Accelerator (H)***

$$H = \{C, BW_{offchip}, BW_{nop}\}$$

Unless otherwise stated, we assume the 2D mesh topology for NoP like Simba [64], and chiplets on two sides (left and right) of the packages have off-chip interfaces.

B. Workload Partitioning Space

To reduce the complexity of the scheduling problem, we adopt a multi-level scheduling method, which splits the end-to-end workload defined in the layer granularity into coarse-grained layer groups, termed as the *time window*. Figure 3 shows an example of the time window that contains six layers from Model A and five layers from Model B.

A time window (tw) is defined by the start time and the duration (T_S and T_{tw}) and a set of assigned layers to the time window, as shown in Definition 4.

Definition 4. ***Time Window (tw)***

For a target workload scenario Sc, a time window tw is defined as follows:

$$tw(Sc) = (T_s, T_{tw}, L)$$

where $L = \{l | l \in Sc\}$

The time window describes a set of layers to be executed on an MCM AI accelerator package, which is used for describing package level scheduling. For each chiplet, we define a finer-grained group of layers within a time window. We term the sub-set of layers within a time window as *segment*.

Definition 5. ***Segment (sg)***

For a time window $tw(Sc)$ and its layers $L(tw(Sc))$, the segment $sg(tw(Sc))$ is defined as follows:

$$sg(tw(Sc)) = \{l | l \in L(tw(Sc))\}$$

To develop a systematic optimization algorithm for layer segmentation i each time window, we need to define the conditions of valid layer segments, provided as follows:

Theorem 1. ***The validity of segments in a time window***

For a time window $tw(Sc)$ and its layers $L(tw(Sc))$, let the set of all segments for $tw(Sc)$ be SG, then SG is valid if the following condition is satisfied:

$$\bigcup_{sg \in SG} sg = L(tw(Sc)) \wedge [\forall sg_i \neq sg_j \in SG, sg_i \cap sg_j = \emptyset]$$

Theorem 1 states two conditions (1) the set of segments needs to cover all the layers in their time window for completing assigned layer computations for the time window and

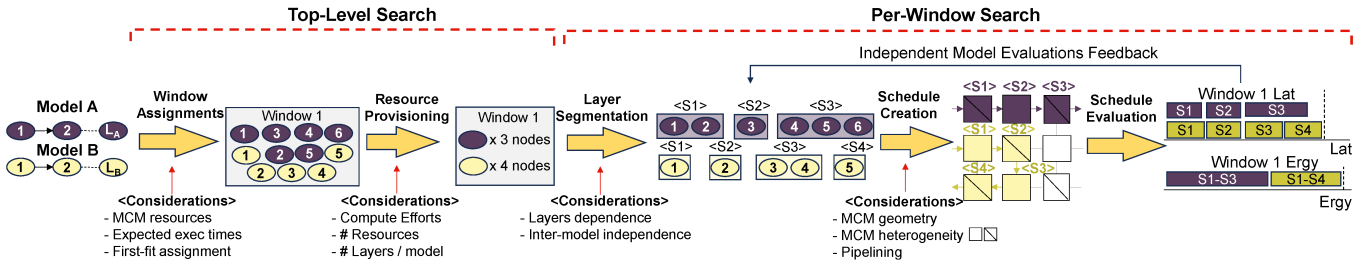

Fig. 3. An overview of our scheduling algorithm for multi-model workloads on heterogeneous MCM AI accelerators.

(2) all segments are exclusive to prevent redundant computing. The same idea extends to the time window as follows:

Theorem 2. ***The validity of time window partitioning***
For a multi-model workload Sc, its layers $L(Sc)$, and the set of time windows $TW(Sc)$, $TW(Sc)$ is valid if the following condition is satisfied:

$$\bigcup_{tw \in TW(Sc)} tw = L(Sc) \wedge [\forall tw_i \neq tw_j \in TW(Sc), tw_i \cap tw_j = \emptyset]$$

Both Theorem 1 and Theorem 2 indicate that the segments and time windows need to be partitions of the workload and time window layers, respectively. Combining all definitions in this section, we formulate the workload partitioning space into the time window and segment as follows:

Definition 6. ***Workload Partitioning Space***
For a multi-model workload Sc, the time window partitioning space ($Sp_{tw}(Sc)$) and the layer segmentation space for a time window ($Sp_{sg}(tw)$) are defined as follows:

$$Sp_{tw}(Sc) = \mathbb{P}(L(Sc))$$
$$Sp_{sg}(tw) = \mathbb{P}(L(tw))$$

where $\mathbb{P}(A)$ refers to all possible partitioning of a set A

C. Scheduling Space

A segment contains layers to be executed on a chiplet. Therefore, spatial (i.e., which segment runs on which chiplet) and temporal mappings (i.e., execution order of segments on each chiplet) of segments construct the scheduling space within each time window when segments are determined. Therefore, the scheduling space within a time window $tw(Sc)$ can be defined as follows:

Definition 7. ***Scheduling Space in a Time Window (SS_{TW})***
For a given time window $tw(Sc)$ and a target MCM accelerator hardware H, the scheduling space within the time window ($SS_{TW}(tw(Sc), H)$ is defined as follows:

$$SS_{TW}(tw(Sc), SG, H)$$
$$= \{(sg, c, j) | sg \in SG \wedge c \in C_H \wedge j \in \mathbb{N} \wedge val(sg, tw(Sc))\}$$

where C_H refers to the set of chiplets in H and val(sg, $tw(Sc)$) indicates the validity of sg for $tw(Sc)$

Each entry in SS_{TW} describes the spatial and temporal mapping of a segment (sg). Spatial mapping can be defined as the target chiplet to execute sg. Accordingly, a target chiplet (c) is specified for sg. The temporal mapping is defined as the execution order. Therefore, a natural number j is used to represent the execution order. Note that the execution order is defined separately on each chiplet. Based on Definition 7, we can define the entire scheduling space as the collection of that in each time window.

Definition 8. ***MCM Scheduling Space for a Multi-model Workload ($SS_{Sc}(H)$)***
For an MCM AI accelerator (H) and a multi-model workload (Sc), the scheduling space ($SS_{Sc}(H)$) is defined as follows:

$$SS_{Sc}(H) = \{(TW, SG_{TW}, (SS_{TW}(tw, SG_{TW}(tw), H)) | TW \subset Sp_{tw}(Sc) \wedge SG_{TW}(tw) \subset Sp_{sg}(tw) \wedge tw \in TW\}$$

where SG_{TW} refers to the set of layer segments for each time window in TW

Definition 8 defines the entire scheduling space of an MCM AI accelerator for a multi-model workload as the cross-product of all possible time window partitioning, layer segmentation for each time window, and corresponding scheduling space within each time window.

D. Scheduling Problem

Based on Definition 8, we define a schedule instance as the collection of spatial and temporal mapping for given valid time windows (TW) and segments for each time window (SG_{TW}).

Definition 9. ***MCM Schedule***
A schedule instance ($sched(Sc, H)$) is defined as follows:

$$sched(Sc, TW, SG_{TW}, H) = \{(TW, SG_{TW}, s) | valid(TW, Sc) \wedge \forall tw \in TW : val(SG_{TW}(tw), tw) \wedge s \in SS_{TW}(tw, SG_{TW}(tw), H)\}$$

where SG_{TW} refers to the set of layer segments for each time window in TW

Using Definition 9, we formulate the scheduling problem as a minimization problem of an optimization metric of choice (e.g., latency and energy), as follows:

Definition 10. ***MCM Scheduling Problem***

$$\underset{TW, SG_{TW}, Sched}{\text{argmin}} \quad OptMetric(TW, SG_{TW}, Sched, H)$$

where $Sched = sched(Sc, TW, SG_{TW}, H)$

The optimization metric can be chosen by users depending on the use case. In our scheduler, we adopt a comprehensive and customizable score that thoroughly considers all of latency, energy, and energy-delay product (EDP), allowing users to configure their own optimization metrics, which can be the mentioned frequently used metrics, or a user-defined function that takes a schedule instance and generates a custom metric.

E. Performance Modeling

In order to evaluate schedules on target MCM AI accelerator hardware, we extend MAESTRO [35] to the chiplet domain and model the latency experienced on MCM AI accelerators when concurrently executing multi-model workloads We discuss our latency evaluation methodology in detail next.

Layer Latency. The latency incurred by an individual layer, l, mapped onto an accelerator chiplet is defined as:

$$Lat(l) = Lat^{ip\_com}(l) + Lat^{comp}(l) + Lat^{op\_com}(l)$$

$Lat^{comp}(l)$ is the layer computation cost dependent on chiplet parameters in Definition 2; $Lat^{ip\_com}(l)$ is latency incurred from loading the layer operands (input activations and weights); $Lat^{op\_com}(l)$ is transmission latency of output activation to a subsequent layer. Lat^{com} is defined as:

$$Lat^{com} = \begin{cases} 0, \text{if same chiplet} \\ \frac{Sz_{data}}{BW_{nop}} + n_{hops} \times Lat_{hop} + \delta, \text{if same package} \\ \frac{Sz_{data}}{BW_{mem}} + n_{hops} \text{ x } Lat_{hop} + Lat_{mem} + \delta, \text{if offhcip} \end{cases}$$

where communication costs are incurred when transmitting data to/from another chiplet on package or the offchip memory. The first term $\frac{Sz_{data}}{BW}$ reflects transmission latency; the second term is captures propagation latency across n_{hops} between the source and destination; δ is an additional latency term for potential NoP traffic conflicts; Lat_{mem} is the offchip memory read/write latency based on memory bandwidth.

Time Window Latency. We first model a layer segment's latency in a time window as follows:

$$Lat(sg) = \sum_{n=1}^{N} Lat^{comp}(l_n) + Lat^{ip\_com}(sg) + Lat^{op\_com}(sg)$$

The first term represents the sum of individual layer computational latencies; Lat^{ip_com} is the initial external off-chiplet data transfer to load inputs and weights; Lat^{op_com} is the transmission latency from transmitting segment output data to the next segment or offchip memory. From here, we can define the time window latency as:

$$Lat(tw) = \max_{SG_m \subset SG} [Lat(SG_m)]$$

where for a time window's assigned set of segments, SG, the overall window latency is the maximum latency incurred by a subset of segments, $SG_m \in SG$, associated with a model m and necessitating sequential execution. $Lat(SG_m)$ can be estimated as the pipelining latency across segments – that is, given model m requires processing data with batch size, b,

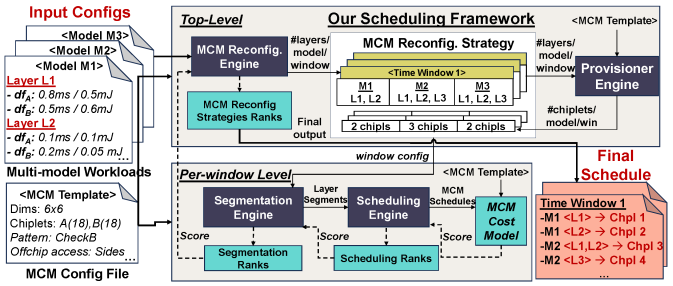

Fig. 4. Our proposed multi-model scheduling framework on heterog. MCM.

and the max number of samples any chiplet can process at a time is given by mini-batch size, $b' <= b$, we get:

$$Lat(SG_m) = \sum_{sg_k \in SG_m} Lat(sg_k|b') + (\frac{b}{b'} - 1) \times \max_{sg_k} [Lat(sg_k|b')]$$

Overall Latency. The overall Scenario latency can then be estimated as the aggregate across all time windows:

$$Lat(Sc) = \sum_{tw_j \in TW} Lat(tw_j)$$

Energy Modeling. Albeit similar to latency, energy costs are always aggregated considering communication energy costs of data sizes, n_{hops}, bit transmission energy, and memory access.

IV. Proposed Scheduling Framework: SCAR

We discuss our scheduling framework, SCAR, for multi-model workloads on heterogeneous MCMs based on the hierarchical search space characterization and problem formulation in Section III. As illustrated in Figure 3, our scheduling algorithm is a two-level approach: top-level and per-window searches. Top-level search is responsible for selecting layers in each model to be scheduled within a time window and determining the initial number of chiplet nodes for each model. Per-window search explores the spatial and temporal partitioning (tiles or layer segments) of the layers in each model at the chiplet granularity. To explore the chiplet granularity tiling space, we generate valid inter-chiplet-pipelined schedules utilizing a scheduling tree structure inspired by the RA Tree [8]. Each schedule is evaluated using our custom heterogeneous MCM cost model which provides feedback to the chiplet level tiling ("layer segmentation" in Figure 3) with expected metrics (latency, energy, EDP, etc.).

We codify our scheduling algorithm into a software framework, as illustrated in Figure 4. As inputs, our scheduling framework receives (1) description files of the multi-model workloads (layer parameters, topology, dependencies, etc.) and (2) a description file of the MCM hardware specification (the number of chiplets, the shape, and chiplet arrays dataflow organization, NoP bandwidth, on-chiplet memory size, etc.). As outputs, our scheduling framework reports an optimized schedule with expected metrics such as latency, energy, EDP, or other user-defined metrics as a combination of latency and energy. Our scheduling framework consists of four software *engines* illustrated in Figure 3 discussed next.

A. MCM Reconfiguration Engine (MCM-Reconfig)

The *MCM-Reconfig* engine at the top-level step receives the multi-model workload descriptions with layer information in each model, layer dependency, and expected latency and energy of each layer on each chiplet class offline-analyzed by MAESTRO [35]. The *MCM-Reconfig* engine is responsible for the window assignment in Figure 3, which (1) generates candidate time window partitioning strategies via sampling a set of discrete points in time as boundary points, and (2) assigns layers from models to each time window. As the final assignment of layers to chiplets is not known apriori, the decisions in *MCM-Reconfig* engine are based on expected execution times. Formally, given $|DF|$ dataflow style classes, the expected execution latency for a layer l is:

$$\mathbb{E}(Lat(l)) = \sum_{i=1}^{|DF|} \frac{n_{df_i}}{|\mathcal{C}|} \times Lat(l \rightarrow i) \quad (1)$$

where n_{df_i} indicates the number of class i chiplets integrated onto the MCM having $|C|$ chiplets in total; $Lat_{l \rightarrow i}$ is layer l latency when scheduled on the class i chiplet, which is retrieved offline from latency database generated by MAESTRO [35], [36]. The average execution time information is utilized in *MCM-Reconfig* engine for window assignment process illustrated in Figure 3.

Time Windows Characterization. *MCM-Reconfig* engine first specifies the number of windows, through a hyperparameter, n_{splits}, to explore proper cut points for each model. For example, in Figure 3, the model A has a cut after layer 6, which led to having layers 1-6 in Window 1. The worst-case latency experienced by any model is set as the time horizon to be partitioned into periodic time windows.

Greedy Layer Packing Algorithm. We adopt a *first-fit greedy-packing* heuristic to assign layers to execution time windows if their execution time is expected to finish within the time window boundaries (see Algorithm 1). Any layer whose execution time lies across two time windows is deferred to the next time window. Through this approach, we enable (i) running low-latency layers in earlier windows (restricts starvation). (ii) dynamically controlling the number of time windows by skipping trivial time windows with no workloads.

Based on our analysis of the periodic window characterization with greedy layer packing using a workload of UNet and GPT2-L against a layer-optimal approach. We found the rate of EDP improvement stagnated after 4 splits. We set n_{splits}=4 (5 time windows) as our default unless otherwise stated.

B. Provisioner Engine (PROV)

The *PROV* engine provides an initial estimate on the number of chiplet needed by each model workload in every time window from a candidate partitioning strategy. PROV assignments are agnostic to the underlying chiplets' properties (dataflow, resources), and hence we refer to chiplets in this state as *nodes*. We implement the *PROV* engine to support exhaustive search or rule-based node distribution assignments. A uniform distribution rule allocates N_i nodes to the i^{th} model as follows:

Algorithm 1 Greedy Layer Packing Algorithm

Input: M (workloads), $\mathcal{T}$, $\mathcal{C}$, DF
Output: $L2W$ (Layer(s) to windows assignments)

Function LAYERASSIGNMENT(M, $\mathcal{C}$, $\mathcal{T}$)
  for $m \in M$ **do**
    $exec\_win = ()$
    $win\_idx, used\_cycles = 0, 0$
    for $l \in m$ **do**
      $\mathbb{E}(Lat(l)) = \sum_{i=1}^{|DF|} \frac{n_{df_i}}{|\mathcal{C}|} \times Lat(l \rightarrow i)$
      while True **do**
        if $win\_idx == |\mathcal{T}|$ **then**
          $Slack$ = $None$
        else
          $Slack$ = $\rho[win\_idx] - used\_cycles$
        if $Slack$ == $None$ or $\mathbb{E}(Lat(l)) <= Slack$ **then**
          $exec\_win$ += $(l,)$
          $used\_cycles$ += $\mathbb{E}(Lat(l))$
          Break
        else
          $L2W[win\_idx][m] = exec\_win$
          $used\_cycles = \mathcal{T}[win\_idx]$
          $exec\_win = ()$
          win_idx += 1
    $L2W[win\_idx][m] = exec\_win$

$$N_i = round(\frac{\mathbb{E}(\mathcal{P}_i)}{\sum_j (\mathbb{E}(\mathcal{P}_j)} \times |\mathcal{C}|) \quad (2)$$

where $\mathbb{E}(\mathcal{P}_i)$ represents the expected value of a target performance optimization metric (latency, energy, EDP) for the model i. $\mathbb{E}(\mathcal{P}_i)$ is computed in a manner similar to the expectation formula in Equation (1).

We ensure every model in the time window is assigned at least one node to progress its execution. The rules enable trading off search complexity for coverage. We analyze the efficacy of the uniform distribution compared to the exhaustive search in Section V.

C. Segmentation Engine (SEG)

The *SEG* module is instantiated every time window to partition topologically sorted model layers into layer segments (Definition 5) that are mappable to computing nodes for exclusive execution throughout the time window. Different segmentation choices reflect various trade-off points between the *layer-sqeuencing* and *layer-pipelining* features: the former concerns with execution locality on the same node; the latter specifies inter-layer and -chiplet pipelining opportunities.

Segmentation Search Space. A segmentation candidate is represented by a sequence of splitting points. Candidate splitting points for a model can be specified after each layer provided to the *SEG*. Given $|L_i|$ and $|N_i|$ as the respective number of layers and number of assigned nodes from the *PROV* to model workload m_i, the max number of segments that can be generated for m_i is upper bounded by N_i. Thus, the overall segmentation space complexity becomes $\mathcal{O}(\Pi_i \binom{L_i}{N_i - 1})$, We incorporate the following heuristics to manage complexity.

Heuristic 1. Product to summation reduction. We reduce complexity by leveraging the independence of segments from different models to divide the search into a two-step process:

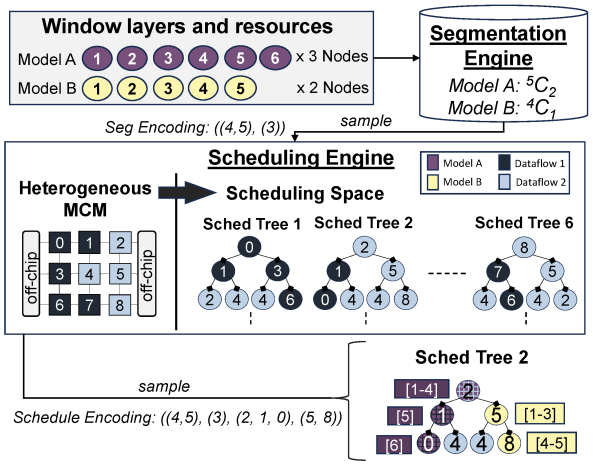

Fig. 5. Schedules creation through the *SEG* and *SCHED* engines.

(1) *SEG* first evaluates segmentation candidates from each model *separately*. (2) The top-k segmentation candidates from each model are used to construct a smaller combinatorial segmentation search space used in the final evlauation. Through this heuristic, the search space complexity is reduced from $\mathcal{O}(\Pi_i \binom{L_i}{N_i-1})$ to $\mathcal{O}(max(\binom{L_i}{N_i-1}))$.

Heuristic 2. Node allocation constraint. To further reduce the search complexity within the *SEG* engine, SCAR supports having a node allocation constraint as a hyperparameter, particularly beneficial in cases where time windows have model workloads with dissimilar layer distributions (e.g., one model with a heavy layer, another with numerous small layers), which could lead to an explosion in the number of trivial segmentation options from the low-cost layers, causing the the *SEG* search space complexity to rise. As such small layers are more suited for continuous execution on the same resource by virtue of their lower computational footprints, and to limit unnecessary, costly inter-chiplet data movements, we designate a node allocation constraint to enable alleviating this added complexity by restricting the number of nodes assigned to workloads with disproportionately large number of layers.

D. Scheduling Engine (SCHED)

The innermost *SCHED* engine is responsible for generating the actual physical mapping of layer segments onto chiplets.

SCHED Search Space. As illustrated in Figure 5, the scheduling search space for the mapping of M model workloads onto $\mathcal{C}$ chiplets is represented by a forest of scheduling trees. We define (i) *forest*; as the entire collection of search trees. (ii) *tree*; a single scheduling tree with all models. (iii) *subtree*; a subset tree exclusively associated with a model.

Scheduling Tree Composition. every node in a scheduling tree corresponds to a unique chiplet resource on the MCM showcasing its distinctive heterogeneous features (i.e., dataflow). Tree edges are constructed based on each chiplet's neighbors connected directly through an interposer. Though a node j can be replicated throughout the tree, it can only be visited once, indicating its exclusive occupancy by a model.

Trees Distinction. within each tree, the root nodes of the subtrees specify different chiplets as potential starting positions for candidate model schedules (see Figure 5). Thus, the scheduling space coverage starts by selecting a tree, represented by a permutation sequence of subtrees' root nodes – e.g., permutation sequence [i,j,k] indicates exploring scheduling candidates for a tree with scheduling candidates starting at chiplet positions i, j, and k for a 3-model workload. The depth of model i's subtree is determined by N_i.

Candidate Schedules Generation. Through traversing each subtree, we can obtain candidate execution schedules for each model by assigning segments orderly to the subtree's nodes. Starting from the root node of the first model's subtree, a constrained depth first search (DFS) is performed generating a candidate schedule path once the full subtree depth (N_i) is reached. This traversal is repeated for each subsequent subtree, constrained on the preceding subtree's prior visited nodes.

Encoding and Search Algorithm. As shown in Figure 5, use a $2 \times |M|$-length tuple to represent the final scheduling encoding, where the first M entries reflect segmentation decisions for each model m_i, and the latter $|M|$ entries reflect schedule mappings of segments to chiplets for each workload. This encoding enables various search strategies (e.g., evolutionary).

Search Space Complexity. Given $|M|$ as the number of models in a given window, $|T|$ the number of scheduling trees in the search space, d is a traversal path's degree of freedom, and N_{max} representing the max number of resources allocated to any model in this window. The scheduling search complexity can be given by $\mathcal{O}(|M| \times |T| \times d^{N_{max}})$.

TABLE II
MCM MICROARCHITECTURE PARAMETERS FROM [55], [64]. ALL NUMBERS ARE SCALED TO 28 NM TECHNOLOGY.

Offchip Memory	DRAM latency	200 ns
	DRAM energy	14.8 pJ/bit
	DRAM bandwidth	64 GB/s
Package	NoP intercon. latency	35 ns/hop
	NoP intercon. energy	2.04 pJ/bit
	NoP intercon. bandwidth	100 GB/s/Chiplet

E. Cost Model and Scoring

We implement a cost model for evaluating scheduling candidates on different performance efficiency metrics.

Cost Model. We build our analytical cost model on top of MAESTRO [35], [36], which is a standardized analytical cost model tool for modeling performance of AI operators' mappings (dataflow and tiling) on AI accelerators, with a reported 96% accuracy of performance estimation compared to low-level RTL simulation. MAESTRO natively enables *intra-chiplet* performance modeling, and we extend it for the MCM AI accelerator setting by integrating the MCM design parameters from Simba [64] as shown in Table II, where we add MCM-specific layers upon MAESTRO leveraging its proven communication overhead modeling methodology to model *offchip* and *inter-chiplet* communication costs. The overall cost model follows the performance models in Section III-E.

Scoring. Scores are estimated based on latency, energy, or EDP metrics following the characterization in Section III. The *SCHED* aggregates scores for each model's schedule, and returns the top performing configuration to the *SEG* engine to

TABLE III

OUR EXPERIMENTAL MULTI-MODEL WORKLOAD SCENARIOS FOR DATACENTER AND AR/VR USE-CASES INSPIRED BY MLPERF [49], [62] AND XRBENCH [38] BENCHMARKS. 'SL' INDICATES SEQUENCE LENGTH

Use-Case	Scenario	Models	Batch Size
Datacenter (MLPerf) [49], [62]	(1) LMs	GPT-L [60] (sl=128) BERT-L [15] (sl=128)	1 3
	(2) LMs + Image	GPT-L [60] (sl=128) BERT-L [15] (sl=128) ResNet-50 [24] (224×224×3)	1 3 1
	(3) LMs + Image	GPT-L [60] (sl=128) BERT-L [15] (sl=128) ResNet-50 [24] (224×224×3)	1 3 32
	(4) LMs + Segmentation + Image	GPT-L [60] (sl=128) BERT-L [15] (sl=128) U-Net [63] (512×512×1) ResNet-50 [24] (224×224×3)	8 24 1 32
	(5) LMs + Segmentation + Image	GPT-L [60] (sl=128) BERT-L [15] (sl=128) BERT-base [15] (sl=128) U-Net [63] (512×512×1) ResNet-50 [24] (224×224×3) GoogleNet [67] (224×224×3)	8 24 24 1 32 32
AR/VR (XRBench) [38]	(6) AR Assistant	D2GO [46] (Object Det.) PlaneRCNN [41] (Plane Det.) MiDaS [61] (Depth Est.) Emformer [66] (Speech Rec.) HRViT [17] (Semantic Seg.)	10 15 30 3 10
	(7) AR Gaming	PlaneRCNN [41] (Plane Det.) Hand S/P [20] (Hand Track.) MiDaS [61] (Depth Est.)	15 45 30
	(8) Outdoors	D2GO [46] (Object Det.) Emformer [66] (Speech Rec.)	30 3
	(9) Social	EyeCod [75] (Gaze Est.) Hand S/P [20] (Hand Track.) Sp2Dense [44] (Depth Ref.)	60 30 30
	(10) VR Gaming	EyeCod [75] (Gaze Est.) Hand S/P [20] (Hand Track.)	60 45

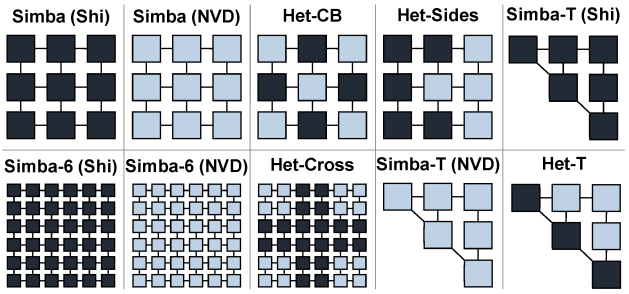

Fig. 6. Evaluated MCM chiplet organizations in this work.

rank segmentation strategies. Top segmentation strategies in each window are aggregated to score the overall scheduling strategy at *MCM-Reconfig* (see the scoring flow in Figure 4).

V. EVALUATION

A. Experimental Settings

Multi-Model Workloads. Our evaluations are performed on multi-model workload scenarios based on models from (1) MLPerf inference benchmark [49], [62], which are curated for datacenter multi-tenancy setting following data center usage trends in [23], [29], [58]. (2) XRBench [38] for multi-model AR/VR workloads. The full list of scenarios is provided in Table III covering a wide range of use-cases with varying degrees of diversity and complexity.

MCM System. We evaluate SCAR on a variety of MCM systems following Simba architecture [64]. Simba comprises a

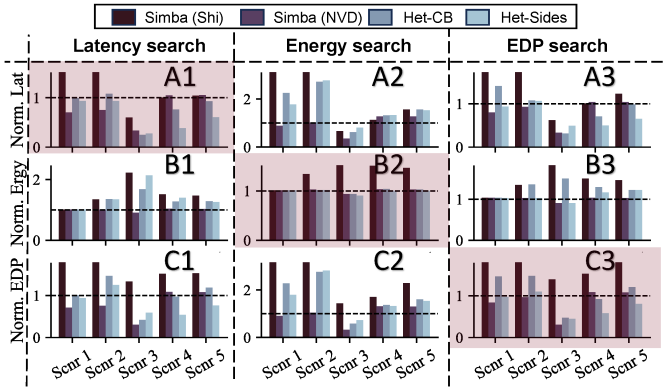

Fig. 7. Latency, energy, and EDP evaluations for top-scoring 3×3 candidates normalized by the standalone NVDLA for datacenter scenarios. Highlighted barplots indicate main results (aligned optimization and evaluation metric).

total of 36 chiplets arranged as four 3×3 groups of chiplets and connected in a Mesh topology. We implement (1) 3×3, and (2) 6×6 MCM templates for our experiments. we adopt XY routing for on-pacakge data movement, and integrate further memory interfaces on the sides of the outer chiplets for direct access to the offchip DRAM as in [19]. We consider 4096 PEs/chiplet and 256 PEs/chiplet for the datacenter and AR/VR settings, respectively. We set L2 shared memory size in each chiplet to 10 MB as in a recent mobile accelerator's on-chip memory size [59].

Baselines and MCM patterns. We choose Shidiannao [16] and NVDLA [52] dataflow styles for our accelerator chiplets given their proven superiority [37] and adopt two baselines:

- *Standalone*. each model is assigned a single accelerator chiplet - all chiplets adopt the same dataflow.
- *Simba-like Pipelining*. In each time window, model workloads can be assigned to more than one chiplet. All chiplets adopt the same dataflow.

Figure 6 illustrates the MCM chiplet patterns evaluated through SCAR. The 3×3 MCM patterns are the default.

Optimization Targets. We perform our search space exploration experiments to target optimizing a single metric at a time, coining the terms Latency Search, Energy Search, and EDP Search. EDP Search is our default experiment.

Search Criteria. We adopt an exhaustive brute-force search for all 3×3 MCM experiments. For the 6×6 experiment, we implement an evolutionary algorithm for the SEG module to navigate the rising complexity. We set the population size and max number of generations to 10 and 4, respectively.

B. Datacenter multi-model scheduling results

We discuss the full datacenter scheduling results on the 3×3 MCM, provided in Table IV and Figure 7. Figure 8 illustrates the Pareto search results for Scenarios 3 and 4.

Impact of Workload Scenarios. We observe that different scenarios are more affine towards different MCM strategies owing to their varying degrees of diversity and computational demands within the multi-model suite. For example in the EDP search in Table IV, scenarios 1-3 favor the standalone (NVD) and Simba (NVD) strategies as both are dominated by low-batched, transformer-based workloads (GPT-L and BERT-L)

TABLE IV

BREAKDOWN OF THE SEARCH RESULTS ACROSS ALL MLPERF DATACENTER SCENARIOS FROM TABLE III. THE COMPARISON INVOLVES THE TOP PERFORMING MODELS ON LATENCY AND EDP FOR EACH SEARCH STRATEGY ACROSS ALL WORKLOAD SCENARIOS. LATENCY ESTINATES AT 500 MHZ.

Strategy	Latency Search										EDP Search									
	Latency (s)					EDP (J.s)					Latency (s)					EDP (J.s)				
	Sc1	Sc2	Sc3	Sc4	Sc5	Sc1	Sc2	Sc3	Sc4	Sc5	Sc1	Sc2	Sc3	Sc4	Sc5	Sc1	Sc2	Sc3	Sc4	Sc5
Stand.(Shi)	1.41	1.41	2.63	14.82	14.82	0.029	0.047	1.21	16.43	21.09	1.41	1.41	2.63	14.82	14.82	0.029	0.047	1.21	16.43	21.0
Stand.(NVD)	0.36	0.36	4.06	7.94	7.94	0.007	0.01	0.786	5.707	7.608	0.36	0.36	4.06	7.94	7.94	**0.007**	**0.01**	0.786	5.707	7.608
Simba (Shi)	0.99	0.99	2.44	7.98	8.29	0.02	0.03	1.048	10.85	11.216	1.13	0.97	2.53	8.03	9.79	0.024	0.024	1.097	8.706	13.75
Simba (NVD)	**0.25**	**0.27**	1.37	8.34	8.34	**0.005**	**0.007**	**0.24**	6.216	8.225	**0.29**	**0.34**	1.36	8.284	8.28	0.006	0.008	**0.238**	6.165	8.162
Het-CB	0.36	0.39	**1.02**	6.04	7.39	0.007	0.013	0.331	5.545	9.051	0.51	0.39	**1.27**	5.6	7.81	0.01	0.013	0.37	5.21	9.166
Het-Sides	0.34	0.34	1.12	**3.07**	**4.81**	0.007	.011	0.463	**3.074**	**5.8**	0.36	0.38	2.0	**3.8**	**5.19**	0.008	0.01	0.35	**3.328**	**6.107**

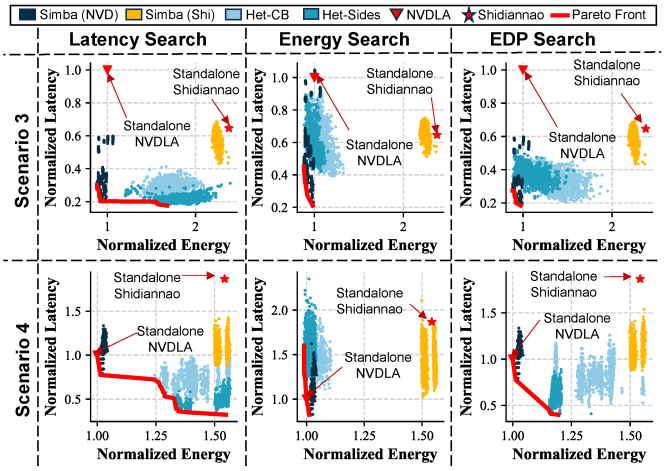

Fig. 8. Pareto results for the brute force search across various MCM strategies on various search targets for scenarios 4 and 5 from Table III.

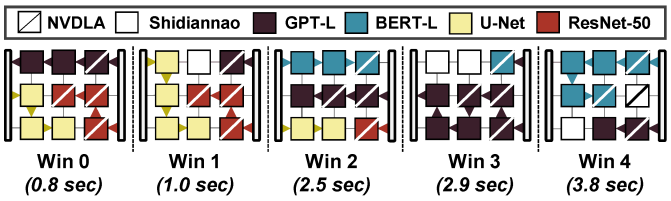

Fig. 9. Top-scoring scheduling strategy for Scenario 4 Het-Sides. Allocation of chiplets and coarse-grained schedules are shown for each time window. Times computed over 500 MHz indicate cumulative window latencies.

characterized by strong affinities towards the NVDLA dataflow style, As the computational load and model diversity increase, schedules obtained through heterogeneous strategies become more favorable. For instance in the EDP search scenarios 4-5 with more batches and heavy ResNet-50 and UNet workloads, Het-Sides outperforms all other strategies on EDP evaluation, experiencing 46.02% and 25.18% less EDP compared to Simba (NVD) on the respective scenarios.

Target Objective Impact. In matching criteria plots (A1, B2, and C3) from Figure 7, Het-Sides schedules for the heaviest scenarios 4-5 outperform schedules from all other strategies. This superiority, however, comes at the expense of other metrics. For instance in C3, the Het-Sides schedule outperforms that from all other strategies by experiencing 0.58× the EDP from the Standalone (NVD). This is achieved, however, by speeding up execution at the expense of energy consumption, where Het-Sides achieves the fastest execution speedup of 2× over Standalone (NVD) in (A3) at the expense of being 1.22× more energy demanding than Standalone (NVD) in (B3).

Pipelining Benefits and NoP impact. Pipelining individually can offer performance improvements over standalone baselines when ample resources are available to the models. For example, Scenario 3 (Latency Search) – top-left most Pareto in Figure 8 – shows Simba (NVD) schedules achieving considerable speedups over the the standalone NVDLA baseline (2.9 × for the top-performing candidate from Table IV). The reason being that as the scheduler targets latency optimization, it attempts to consistently identify chiplet assignment strategies that equate pipelining latencies between model workloads assigned to the same time window. For instance, in the last time window containing ResNet-50, the Simba (NVD) scheduler for Scenario 3 queued 18, 14, and 29 layers from the GPT-L, BERT-L, and ResNet-50 for processing, provisioned 3 chiplets for each subset, and yielded accordingly 3 layer segments per model. This configuration led to the model workloads achieving comparable inter-chiplet pipelining latencies of 0.28 ms, 0.32, and 0.33 ms, eventually leading the full schedule to sustain a total of 1.37 s latency compared to the 4.06 s from the standalone NVDLA. We also observe that keeping medium-sized data traffic on package through chiplet-to-chiplet data passing contributes additional energy savings for these medium sized workloads by decreasing offchip memory read/write accesses at the segments' start and end times.

For the NoP traffic, Figure 7-A1 summarizes our findings. In Scenarios 1-3, Simba (NVD) leverages inter-chiplet pipelining to outweigh added NoP costs, achieving latency speedups over standalone NVDLA reaching 1.4×, 1.3×, and 2.9×, respectively. In Scenarios 4-5, Simba (NVD) to incur 1.05× slowdown from the added traffic in both scenarios.

Heterogeneity Benefits. Heterogeneous integration patterns compensate for the added NoP traffic in the heavier workloads (Scenarios 4-5) and boost efficiency through considering per-layer dataflow affinities. Het-Sides achieves 1.7× and 1.25× EDP efficiency over the standalone (NVD). We find also that Het-Sides is always superior to Het-CB in such heavy scenarios as it offers opportunities for both homogeneous and heterogeneous inter-chiplet pipelining. This is beneficial for batched layer sequences with same dataflow affinities.

Het-Sides Top Schedule. In Figure 9, we illustrate the top-scoring Het-Sides schedule from the EDP search in Scenario 4. As shown, the greedy-packing algorithm leads to non-uniform windows where smaller workloads (ResNet-50) are assigned to the earlier windows at the expense of larger workloads (BERT-

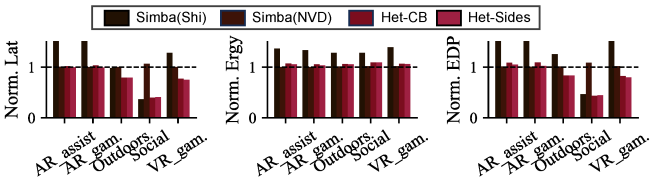

Fig. 10. Evaluations on the EDP search for the XRBench usage scenarios listed in Table 1 normalized by NVDLA standalone configuration.

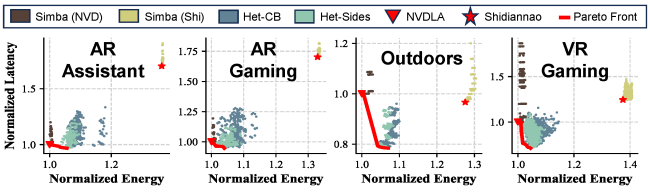

Fig. 11. Pareto optimal results on the EDP search experiments for the labeled XRBench usage scenarios. Results normalized by standalone NVDLA

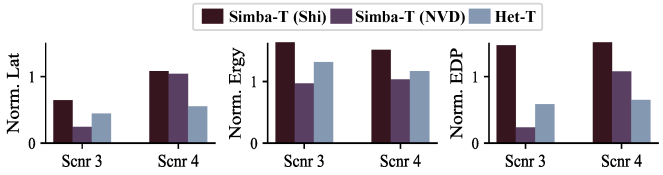

Fig. 12. Results for the EDP Search for Scenarios 3 and 4 on the Triangular NoP topologies from Figure 6 (Normalized by standalone NVDLA).

L). This facilitates (i) fine grained optimization of small workload schedules; (ii) avoiding small workloads starvation. GPT-L and BERT workloads dominate the schedule from window 2. Table VI breaks down the latency in each window.

C. AR/VR multi-model scheduling results

The AR/VR scheduling results on the EDP search for the 3×3 MCM are provided in Table V and Figure 10. Figure 11 illustrates their the Pareto search results. On average, the Het-Sides schedule achieves 17% and 21.3% EDP improvement over standalone NVDLA and Simba (NVD), respectively. We also find that the smaller MCM system limits improvement potential for the heaviest scenarios, AR assistant (scenario 6) and AR gaming (scenario 7) due to increased resource contention, leading the heterogeneous and NVDLA based schedules to achieve comparable evaluations (see Figure 10).

D. Scaling to 6×6 MCM system

We assess how SCAR scheduling scales with the hardware using the 6×6 full Simba MCM system with its templates from Figure 6. We implement an evolutionary algorithm for the *SEG* engine to scale with the rising problem size, and show the results for the EDP search on Scenario 4 at n_{splits}=2 and n_{splits}=3. We choose the heterogeneous cross (Het-Cross) as our heterogeneous template following the insights from the 3×3 experiment regarding choosing heterogeneity patterns that enable both homogeneous and heterogeneous pipelining capabilities. As shown in Figure 13, the evolutionary search has led to identifying configurations for Het-Cross that achieve 2.3× and 1.9× reduction in EDP; 2.1× and 1.8× reduction in latency over Simba (Shi) and Simba (NVD), respectively.

TABLE V
EDP SEARCH AR/VR RESULTS NORMALIZED BY STANDALONE NVDLA

Strgy	EDP Search									
	Relative Latency					Relative EDP				
	Sc6	Sc7	Sc8	Sc9	Sc10	Sc6	Sc7	Sc8	Sc9	Sc10
Stand.(Shi)	1.7	1.7	0.96	0.36	1.24	2.32	2.26	1.22	0.45	1.71
Stand.(NVD)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Simba(Shi)	1.72	1.75	0.97	0.36	1.28	2.36	2.33	1.24	0.47	1.78
Simba(NVD)	1.01	1.0	0.99	1.07	1.0	1.01	1.01	1.0	1.08	1.02
Het-CB	1.01	1.03	0.78	0.4	0.77	1.09	1.09	0.83	0.43	0.82
Het-Sides	1.0	0.99	0.78	0.41	0.75	1.05	1.03	0.83	0.45	0.79

TABLE VI
END-TO-END LATENCY BREAKDOWN IN SECONDS FOR THE TOP PARTITIONING CANDIDATE IN FIGURE 9.

	W0	W1	W2	W3	W4	ideal	tot	#layers
GPT-L	0.23	0.21	1.02	0.28	0.23	1.97	3.1	120
BERT-L	0	0	1.47	0.4	0.90	2.77	3.76	60
U-Net	0.21	0.14	0.46	0	0	0.8	1.45	23
ResNet	0.78	0.17	0.11	0	0	1.1	1.1	66
Window	0.78	0.21	1.47	0.4	0.9	-	3.8	269
#layers	60	30	131	25	23	-	-	

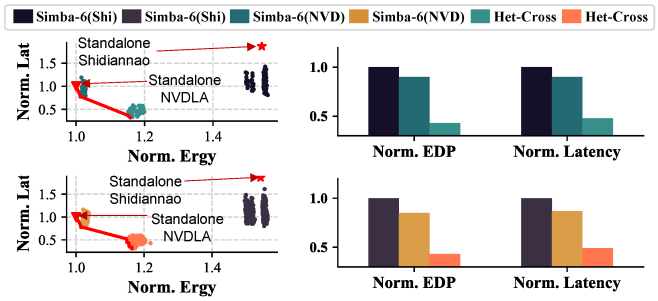

Fig. 13. Pareto Plot and comparison of top performing models for the EDP search on the 6 × 6 MCM at n_{splits}=2 (*top*) and n_{splits}=3 (*bot*).

E. Ablation Studies

Ablation Study on Time Partitioning. Using Scenario 4 and Het-Sides strategy, we study how performance changes when varying n_{splits} from 1 to 5 and repeating the EDP search experiment. Prior to n_{splits}=4, the average rate of reduction in EDP was 1.25×. The rate of improvement drops to only 1.04× between n_{splits}=4 to 5, indicating diminishing returns.
Ablation Study on other NoP Topology. SCAR can generalize to other NoP topologies as it relies on adjacency matrix connectivity. We test this by performing the EDP search for Scenarios 3 and 4 using the triangular NoP topologies in Figure 6. As shown in Figure 12, similar performance patterns are exhibited compared to the Scenario results on the 3×3 Mesh, albeit with varying relative gains. For instance, though Het-T outperforms both Simba-T strategies in Scenario 4 by 2.5× and 1.67× over Simba-T (Shi) and Simba-T (NVD), respectively, it is second best compared to Simba-T (NVD) in Scenario 3 by a factor of 2.5× (compared to 1.47× from Figure 7-C3) due to the increased competition for resources.
Ablation Study on rule-based PROV. We repeated the EDP search for all strategies across Scenarios 3-5 on the 3×3 template using an exhaustive search over the N_i values. Though the results improved further with the added search complexity, the insights remained the same. For Scenarios 4-5, Het-Sides remained superior, achieving respective EDP reductions of 38.3% and 29.9% from Standalone NVDLA; 59.5%

and 57.6% from Simba (Shi); 33.0% and 28.3% from Simba (NVD). For Scenario 3, Simba (NVD) remained superior, achieving 79.7% EDP reduction Het-Sides. These evaluations follow the trends from our rule-based results in Table IV.

Ablation on Greedy Packing Algorithm. Using Scenario 4 and Het-Sides, we test the efficacy of our first-fit greedy layer packing algorithm against a uniform packing baseline, distributing layers uniformly across time windows. Ours achieved 21.8% speedup and 8.6% energy reduction.

F. Summary of Results and Main Insights

We summarize our main insights and findings as follows.

- Heterogeneous MCM patterns improve performance for heavy and diverse multi-model workloads (scenarios 4-5).
- Homogeneous MCM patterns are more suited for small multi-model workloads (scenarios 1-3).
- Heterogeneous MCM patterns with diverse pipelining options (Het-Sides) are superior to heterogeneous patterns with homogeneous pipelining options (Het-CB).
- The target optimization objective is crucial in identifying the best integration strategy. In EDP search scenario 4, Het-sides outperformed all other strategies on EDP, but not on pure energy consumption.
- Topology and number of resources affect the extent of performance improvement for heterogeneous strategies.

Our findings show that understanding multi-model workload characteristics and usage scenarios is crucial for identifying the best MCM integration strategy for a target objective.

VI. Discussion and Limitations

Multi-model optimization targets. We experimented with different optimization targets (latency, EDP, energy) for our scenarios, and showed that the top performing strategy can change based on the target objective. As multi-model workloads evolve, it may be desirable to assign separate optimization targets for different models within a scenario (EDP v. lat). One practical way to achieve this in our framework is by adding a constraint in our EDP search, invalidating schedules that have certain models violate a latency constraint (i.e., the EDP search becomes lower bounded by the latency search).

Heterogeneous chiplets technology. Heterogeneous chiplet integration has become a viable, cost-effective approach to design state-of-the-art AI systems. Nvidia's world-class superchips are a successful example of heterogeneous on-package integration (e.g., Grace-Blackwell (1 CPU + 2 GPUs) [14]). The success of these systems and others (AMD's MI300X [1]) is testament to the hardware manufacturers' investment in chiplets technology, where through advanced manufacturing processes and heterogeneous integration capabilities, the development of MCM AI accelerators (like Nvidia's Simba [64]) becomes more accessible, allowing chiplet modifications/replacement in MCM hardware at lower costs without requiring a complete overhaul of the entire package.

Scheduler Software Integration. SCAR can be integrated on top of existing compiler infrastructure. The advanced scheduling techniques supported by the scheduler (dynamic chiplet regrouping, inter-chiplet pipelining) represent high-level abstractions of the computational graphs that can be transformed through standard compiler software (e.g., MLIR [39]) to representations suited for the underlying hardware. For example, dynamic chiplets regrouping is correspondent to graph partitioning, where a model's computational graph is divided into smaller subgraphs, each associated with the set of computing nodes assigned during the corresponding time window. The subgraphs can then be transformed to lower representations covering the details of buffer management, die-to-die communication, memory R/W requests, I/O, all the way to the transformations covering the dataflow features (loop reordering, spatial unrolling) for the specialized accelerators.

TABLE VII
Comparison against prior related scheduling works.

Work	Chiplet-based Systems	Multi-Models	Inter-Layer Pipelining	Heterog-Aware
Simba [64]	✓		✓	
Tangram [19]			✓	
NN-baton [68]	✓			
SET [8]			✓	
Gemini [9]	✓		✓	
Herald [37]		✓		✓
MAGMA [32]		✓		✓
Planaria [21]		✓		✓
Veltair [42]		✓		
MoCA [33]		✓		
This Work	✓	✓	✓	✓

VII. Related Works

Scheduler for Accelerators. Table VII compares our work against prior scheduling works. As shown, the related works can be categorized into two groups: one which has considered aspects of inter-layer pipelining and chiplet-based systems [8], [9], [19], [64], [68], and another that focused on multi-model workloads on heterogeneous platforms [21], [32], [33], [37], [42]. Only this work addressed MCM, multi-model workloads, inter-layer pipelining, and heterogeneous dataflow.

Multi-chiplet Modules. Several works proposed to address the scalability challenge for DNN acceleration via MCM integration [3], [28], [55], [64], [68]. Most notably, Simba [64] pioneered a scalable deep learning MCM inference accelerator leveraging non-uniform work partitioning, communication-aware data placement, and cross-layer pipelining.

Intra- and Inter-layer Parallelism. Prior works explored intra-layer parallelism to maximize DNN performance efficiency by partitioning DNN layers into smaller, parallelizable tiles [25]–[27], [43], [57], [73], [74]. Other works studied the inter-layer scheduling space to compensate for workloads with low degrees of parallelism [6], [8], [19], [31], [45], [53], [78].

VIII. Conclusion

In this work, we explored the scheduling space of a new class of MCM accelerator architecture, heterogeneous MCM AI accelerator, targeting multi-model AI workloads. We identify that the scheduling problem is intractably large but multi-level problem formulation and heuristics we proposed are effective for the large-scale scheduling problem. The results also show that heterogeneous MCM accelerator is beneficial for multi-model workloads, which motivates further exploration.

REFERENCES

[1] AMD. Amd instinct mi300x accelerator. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/data-sheets/amd-instinct-mi300x-data-sheet.pdf.

[2] Apple. Apple Vision Pro Specs. https://www.apple.com/apple-vision-pro/specs/, 2024.

[3] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. Mcm-gpu: Multi-chip-module gpus for continued performance scalability. *ACM SIGARCH Computer Architecture News*, 45(2):320–332, 2017.

[4] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 940–953. IEEE, 2020.

[5] Noah Beck, Sean White, Milam Paraschou, and Samuel Naffziger. 'zeppelin': An soc for multichip architectures. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 40–42. IEEE, 2018.

[6] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–17, 2023.

[9] Jingwei Cai, Zuotong Wu, Sen Peng, Yuchen Wei, Zhanhong Tan, Guiming Shi, Mingyu Gao, and Kaisheng Ma. Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 156–171. IEEE, 2024.

[10] Prasanth Chatarasi, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. Marvel: a data-centric approach for mapping deep learning operators on spatial accelerators. *ACM Transactions on Architecture and Code Optimization (TACO)*, 19(1):1–26, 2021.

[11] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Multi-model machine learning inference serving with gpu spatial partitioning. *arXiv preprint arXiv:2109.01611*, 2021.

[12] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Serving heterogeneous machine learning models on {Multi-GPU} servers with {Spatio-Temporal} sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 199–216, 2022.

[13] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A {Low-Latency} online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.

[14] CRN. Nvidia Reveals Next-Gen Blackwell GPUs, Promised To 'Unlock Breakthroughs' In GenAI. https://www.crn.com/news/components-peripherals/2024/nvidia-reveals-next-gen-blackwell-gpus-promised-to-unlock-breakthroughs-in-genai, 2024.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[17] FacebookResearch. Hrvit-b1. https://github.com/facebookresearch/HRViT/blob/main/models/hrvit.py#L1125-L1155, 2022.

[18] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, et al. Mtia: First generation silicon targeting meta's recommendation systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–13, 2023.

[19] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 807–820, 2019.

[20] Liuhao Ge, Zhou Ren, Yuncheng Li, Zehao Xue, Yingying Wang, Jianfei Cai, and Junsong Yuan. 3d hand shape and pose estimation from a single rgb image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10833–10842, 2019.

[21] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 681–697. IEEE, 2020.

[22] Johann Hauswald, Yiping Kang, Michael A Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G Dreslinski, Jason Mars, and Lingjia Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. *ACM SIGARCH Computer Architecture News*, 43(3S):27–40, 2015.

[23] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 943–958, 2021.

[26] Charles Hong, Qijing Huang, Grace Dinh, Mahesh Subedar, and Yakun Sophia Shao. Dosa: Differentiable model-based one-loop search for dnn accelerators. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 209–224, 2023.

[27] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. Cosa: Scheduling by constrained optimization for spatial accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 554–566. IEEE, 2021.

[28] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 968–981. IEEE, 2020.

[29] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[30] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H Loh. Enabling interposer-based disintegration of multi-core processors. In *Proceedings of the 48th international symposium on Microarchitecture*, pages 546–558, 2015.

[31] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 622–636. IEEE, 2020.

[32] Sheng-Chun Kao and Tushar Krishna. Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 814–830. IEEE, 2022.

[33] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 828–841. IEEE, 2023.

[34] Seah Kim, Hyoukjun Kwon, Jinook Song, Jihyuck Jo, Yu-Hsin Chen, Liangzhen Lai, and Vikas Chandra. Dream: A dynamic scheduler

for dynamic real-time multi-model ml workloads. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, pages 73–86, 2023.

[35] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 754–768, 2019.

[36] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE micro*, 40(3):20–29, 2020.

[37] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. Heterogeneous dataflow accelerators for multi-dnn workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 71–83. IEEE, 2021.

[38] Hyoukjun Kwon, Krishnakumar Nair, Jamin Seo, Jason Yik, Debabrata Mohapatra, Dongyuan Zhan, Jinook Song, Peter Capak, Peizhao Zhang, Peter Vajda, et al. Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse. *Proceedings of Machine Learning and Systems*, 5, 2023.

[39] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.

[40] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 173–189, 2022.

[41] Chen Liu, Kihwan Kim, Jinwei Gu, Yasutaka Furukawa, and Jan Kautz. Planercnn: 3d plane detection and reconstruction from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4450–4459, 2019.

[42] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. Veltair: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 388–401, 2022.

[43] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. Tenet: A framework for modeling tensor dataflow based on relation-centric notation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 720–733. IEEE, 2021.

[44] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4796–4803. IEEE, 2018.

[45] Xiaohan Ma, Chang Si, Ying Wang, Cheng Liu, and Lei Zhang. Nasa: accelerating neural network design with a nas processor. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 790–803. IEEE, 2021.

[46] Meta. D2go. https://github.com/facebookresearch/d2go, 2022.

[47] Microsoft. Announcing microsoft copilot, your everyday ai companion. https://blogs.microsoft.com/blog/2023/09/21/announcing-microsoft-copilot-your-everyday-ai-companion/, 2023.

[48] Microsoft. Azure openai service. https://azure.microsoft.com/en-us/products/ai-services/openai-service, 2023.

[49] MLCommons. Mlperf inference. https://mlcommons.org/benchmarks/inference-datacenter/, 2023.

[50] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H Loh, Mahesh Subramony, and Sean White. Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 57–70. IEEE, 2021.

[51] Sokratis Nikolaidis, Stylianos I. Venieris, and Iakovos S. Venieris. Multitasc: A multi-tenancy-aware scheduler for cascaded dnn inference at the consumer edge. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 411–416, 2023.

[52] NVIDIA. Nvdla deep learning accelerator. http://nvdla.org,2017., 2023.

[53] Mohanad Odema, Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Magnas: A mapping-aware graph neural architecture search framework for heterogeneous mpsoc deployment. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–26, 2023.

[54] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[55] Marcelo Orenes-Vera, Esin Tureci, David Wentzlaf, and Margaret Martonosi. Massive data-centric parallelism in the chiplet era. *arXiv preprint arXiv:2304.09389*, 2023.

[56] Ioannis Panopoulos, Stylianos Venieris, and Iakovos Venieris. Carin: Constraint-aware and responsive inference on heterogeneous devices for single-and multi-dnn workloads. *ACM Transactions on Embedded Computing Systems*, 2024.

[57] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.

[58] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. *arXiv preprint arXiv:1811.09886*, 2018.

[59] Qualcomm. Quacomm hexagon 680. https://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.24-Monday-Epub/HC27.24.20-Multimedia-Epub/HC27.24.211-Hexagon680-Codrescu-Qualcomm.pdf, 2015.

[60] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[61] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1623–1637, 2020.

[62] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[63] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[64] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–27, 2019.

[65] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 322–337, 2019.

[66] Yangyang Shi, Yongqiang Wang, Chunyang Wu, Ching-Feng Yeh, Julian Chan, Frank Zhang, Duc Le, and Mike Seltzer. Emformer: Efficient memory transformer based acoustic model for low latency streaming speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6783–6787. IEEE, 2021.

[67] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[68] Zhanhong Tan, Hongyu Cai, Runpei Dong, and Kaisheng Ma. Nn-baton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators. In *2021 ACM/IEEE 48th Annual International*

Symposium on Computer Architecture (ISCA), pages 1013–1026. IEEE, 2021.

[69] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[70] Pascal Vivet, Eric Guthmuller, Yvain Thonnart, Gael Pillonnet, Guillaume Moritz, Ivan Miro-Panadès, Cesar Fuguet, Jean Durupt, Christian Bernard, Didier Varreau, Julian Pontes, Sebastien Thuries, David Coriat, Michel Harrand, Denis Dutoit, Didier Lattard, Lucile Arnaud, Jean Charbonnier, Perceval Coudrain, Arnaud Garnier, Frederic Berger, Alain Gueugnot, Alain Greiner, Quentin Meunier, Alexis Farcy, Alexandre Arriordaz, Severine Cheramy, and Fabien Clermidy. 2.3 a 220gops 96-core processor with 6 chiplets 3d-stacked on an active interposer offering 0.6 ns/mm latency, 3tb/s/mm 2 inter-chiplet interconnects and 156mw/mm 2@ 82%-peak-efficiency dc-dc converters. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 46–48. IEEE, 2020.

[71] Zhenyu Wang, Gopikrishnan Raveendran Nair, Gokul Krishnan, Sumit K Mandal, Ninoo Cherian, Jae-Sun Seo, Chaitali Chakrabarti, Umit Y Ogras, and Yu Cao. Ai computing in light of 2.5 d interconnect roadmap: Big-little chiplets for in-memory acceleration. In *2022 International Electron Devices Meeting (IEDM)*, pages 23–6. IEEE, 2022.

[72] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*, pages 331–344. IEEE, 2019.

[73] Yannan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S Emer. Sparseloop: An analytical approach to sparse tensor accelerator modeling. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1377–1395. IEEE, 2022.

[74] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. Hasco: Towards agile hardware and software co-design for tensor computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1055–1068. IEEE, 2021.

[75] Haoran You, Cheng Wan, Yang Zhao, Zhongzhi Yu, Yonggan Fu, Jiayi Yuan, Shang Wu, Shunyao Zhang, Yongan Zhang, Chaojian Li, et al. Eyecod: eye tracking system acceleration via flatcam-based algorithm & accelerator co-design. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 610–622, 2022.

[76] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1049–1062, 2019.

[77] Xinyi Zhang, Cong Hao, Peipei Zhou, Alex Jones, and Jingtong Hu. H2h: heterogeneous model to heterogeneous system mapping with computation and communication awareness. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 601–606, 2022.

[78] Shixuan Zheng, Xianjue Zhang, Leibo Liu, Shaojun Wei, and Shouyi Yin. Atomic dataflow based graph-level workload orchestration for scalable dnn accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 475–489. IEEE, 2022.