Para-Pipe: Exploiting Hierarchical Operator Parallelism of ML Computational Graphs on SoCs

Yujie Zhang, Huiying Lan, Ehsan Aghapour, Zhiyuan Ning,
Peng Zan, Weidong Shao, Anuj Pathania, and Tulika Mitra

***Abstract*—As edge-based deep learning applications become more complex, optimizing performance on heterogeneous System-on-Chips (SoCs) presents unique challenges. Traditional pipelining techniques distributing the computation across different on-chip processing units, while effective for throughput, do not address the latency demands posed by modern neural networks with complex interdependencies and extensive operator parallelism. There is a potential in leveraging operator parallelism to enable concurrent execution across multiple processing units, thereby reducing inference latency. However, prioritizing pipelining or parallel execution often necessitates a compromise, where optimizing one performance metric adversely impacts the other.**

This paper introduces *Para-Pipe*, a hierarchical mapping framework that integrates intra- and inter-stage operator parallelism within a pipelined architecture. *Para-Pipe* navigates the trade-off between throughput and latency by selectively fine-tuning parallelism levels within and across pipeline stages. This strategy can significantly reduce inter-processor communication overhead, significantly improving energy efficiency. Our evaluation demonstrates that *Para-Pipe* generates multiple Pareto-optimal configurations, achieving a balance between throughput and latency on an Amlogic SoC equipped with *ARM big.LITTLE* CPUs and GPU, as well as the Black Sesame Technology SoC featuring a deep learning accelerator and two DSPs. More importantly, throughput-optimized configurations under *Para-Pipe* on Amlogic SoC show an average energy efficiency improvement of 11.0% over purely pipelined strategies and 23.3% relative to non-pipelined parallel execution.

***Index Terms*—Hierarchical operator parallelism, heterogeneous SoCs, latency-throughput trade-off, energy efficiency.**

I. Introduction

EDGE devices increasingly utilize heterogeneous multi-processor System-on-Chips (SoCs), which comprise a variety of computing units, including CPUs, GPUs, and specialized hardware accelerators, each distinguished by unique power-performance characteristics [1]. Despite the capabilities of these diverse units, mainstream machine learning (ML) frameworks such as PyTorch [2], TVM [3], and the *ARM Compute Library (ARM-CL)* [4] frequently underutilize them. Typically, these frameworks leverage the most high-performance or energy-efficient unit for sequential task execution, simplifying software complexity. While this approach facilitates programming and deployment across different neural networks on edge devices, it neglects the advantages of simultaneously utilizing multiple compute units, which could significantly improve execution speed and energy efficiency.

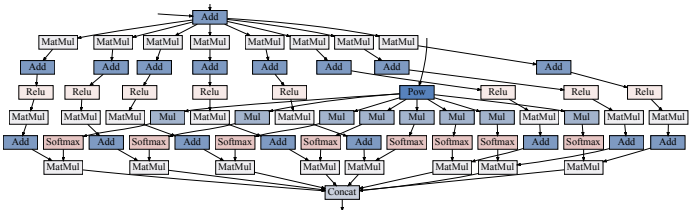


Fig. 1. A portion of PETR network architecture to illustrate intensive operator parallelism [5].

To better harness diverse resources, pipelining techniques partition ML computational graphs and allocate segments to specific compute units on the SoC, effectively maximizing throughput by fully utilizing on-chip resources for concurrent processing of streaming inputs [6]–[11]. These methods are primarily designed for traditional convolutional neural networks like MobileNet. Their predominantly sequential architecture is conducive to straightforward partitioning. However, the growing complexity and inter-dependencies among operators in modern neural networks present substantial challenges for effective pipelining. Consider, for example, PETR network [5], a spatiotemporal transformer-based encoder for autonomous driving perception tasks, with partial architecture illustrated in Fig. 1. Identifying optimal partition points within such complex networks is increasingly challenging due to the expanded number of operations and potential partition points. Moreover, this approach can lead to increased latency per frame due to enhanced inter-pipeline stage communication overhead and suboptimal use of concurrent operator execution within individual frames. Such latency increases are particularly detrimental in real-time decision-making applications, including autonomous driving [12] and some IoT devices [13], [14], where prompt response times are imperative.

Despite their complexity, modern neural networks are ripe with opportunities for concurrent operator execution to improve latency markedly. For instance, the InceptionNet family [15] employs dimension-reduced inception modules that process the same input with filters of varying scales in parallel. Similarly, transformer-based networks utilize multiple attention heads within self-attention mechanisms to parallel-process the same input sequence. These computational designs

This work was supported by the National Research Foundation, Singapore under its Competitive Research Program Award NRF-CRP23-2019-0003 and by a gift from Black Sesame Technologies. This article was recommended by Associate Editor A. Kumar. (*Corresponding author: Tulika Mitra.*)

Yujie Zhang, Huiying Lan, and Tulika Mitra are with the School of Computing, National University of Singapore, Singapore. (e-mail: zyujie@comp.nus.edu.sg; hy.lan@nus.edu.sg; tulika@comp.nus.edu.sg).

Ehsan Aghapour and Anuj Pathania are with Parallel Computer Systems, the University of Amsterdam, 1098 XH Amsterdam, The Netherlands (e-mail: e.aghapour@uva.nl; a.pathania@uva.nl).

Zhiyuan Ning, Peng Zan, and Weidong Shao are with the Information Technology Department, Black Sesame Technologies, San Jose, CA 95131 USA (e-mail: thomas.ning@bst.ai; peng.zan@bst.ai; david.shao@bst.ai).

demonstrate considerable inter-operator parallelism, devoid of complex data dependencies, suggesting that strategic partitioning and allocation of operations to multiple processing units for truly parallel execution could substantially enhance latency. However, prioritizing latency in single-frame acceleration by engaging all available units [16]–[20] undermines the throughput benefits of pipelining.

Achieving a balanced trade-off between latency and throughput is critical for optimizing performance in edge-based deep learning applications. Enhancements in throughput reduce overall execution times for processing large-scale inputs, while latency improvements decrease delays for individual frames. In applications like video surveillance [21]–[23], where both rapid processing of extensive data and swift handling of single frames are essential, optimizing both throughput and latency concurrently is imperative. In these instances, standard pipelining or parallel execution strategies that promote throughput at the expense of latency—or vice versa—are suboptimal.

To tailor and optimize the balance between latency and throughput in neural networks characterized by intensive operator parallelism on heterogeneous SoCs, we introduce *Para-Pipe*. This hybrid framework integrates pipelining with parallel execution through a hierarchical structure. *Para-Pipe* systematically identifies neural network subgraphs suitable for parallel processing (e.g., GoogLeNet inception modules) and allocates them across multiple units to maximize this parallelism. For subgraphs exhibiting a lower operator parallelism, it assigns them to individual units for sequential processing. This dual approach effectively separates sequential and parallel processing into distinct pipeline stages, thereby facilitating hierarchical exploitation of inter-operator parallelism within and across stages. *Para-Pipe* can selectively modulate intra- and inter-stage operator parallelism levels, finely tuning the latency and throughput balance. Correspondingly, it regulates the cost associated with inter-processor communication within and among stages, optimizing for significant reductions and markedly enhancing energy efficiency—a critical issue in embedded heterogeneous computing environments, especially in multiprocessor SoCs [20], [24], [25]. Para-Pipe stands out in terms of two aspects:

Para-Pipe optimizes the latency-throughput trade-off. *Para-Pipe* provides the flexibility to adjust the workload of parallel operator execution, enabling tailored optimization to suit specific computational requirements. Generally, latency decreases when fewer pipeline stages and more parallel computing resources are allocated to each stage, while throughput benefits from a diminished workload per stage by adding more pipeline stages. The lowest latency is achieved by engaging all available processors for single-frame inference without pipelining. Meanwhile, the highest throughput is achieved for multi-frame inference by creating multiple pipeline stages so that the execution time of the longest stage is at a minimum. Crucially, based on the workload customization of parallel operator execution within stages, *Para-Pipe* effectively manages the trade-off between latency and throughput.

Para-Pipe enhances energy efficiency in heterogeneous SoCs. Relative to parallelism-only methods, *Para-Pipe*'s hybrid execution mode significantly reduces inter-processor communication costs by allocating specific subgraphs to stages managed by a single processor for sequential execution. Moreover, compared to the pipeline-only setup, *Para-Pipe* diminishes the necessity for inter-stage communication and coordination by streamlining the number of pipeline stages. Nonetheless, the parallel execution stages are not entirely exempt from incurring inter-processor communication costs. Through the judicious selection of processors for concurrent execution within a single stage, *Para-Pipe* effectively mitigates such overhead, thereby augmenting energy efficiency.

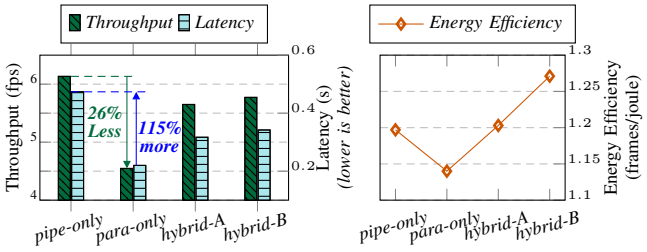


Fig. 2. Latency-throughput trade-offs and energy efficiency for scheduling and mapping options for Inception-v3 [27] on Amlogic SoC [26]. hybrid-A and hybrid-B combine pipelined and parallel execution with varying degrees of parallel operator execution.

We evaluate our framework across four InceptionNet family and two transformer-based models on two real SoCs, an Amlogic A331D SoC [26], equipped with dual CPU clusters and a GPU, in addition to the Black Sesame Technology (BST) SoC including an NPU and two DSPs. The results reveal that *Para-Pipe* generates Pareto-optimal mapping options tailored to various throughput and latency priorities. Notably, hybrid configurations optimized for throughput on Amlogic SoC achieve an average energy efficiency improvement of 11.0% compared to purely pipelined strategies and 23.3% relative to non-pipelined parallel execution.

Motivating Example: Fig. 2 shows performance trade-offs associated with various mapping options during Inception-v3 network [27] inference on an Amlogic SoC. Methods focusing solely on pipelining ('pipe-only') or parallel execution ('para-only') show significant gains in throughput and latency, respectively. However, these approaches also reveal notable degradation in other metrics, a 115% increase in latency for 'pipe-only' and a 26% decrease in throughput for 'para-only'. In contrast, the 'hybrid-A' and 'hybrid-B' strategies, which integrate pipelining with parallel execution, offer a more balanced performance by modulating the degree of inter-operator parallelism within the pipeline stages. Notably, the 'hybrid-A' configuration leads to a 7.9% reduction in throughput but concurrently secures a substantial latency improvement of 33.1%, compared to the 'pipe-only' approach. Furthermore, the hybrid methods enhance the energy efficiency of the SoC; for instance, the 'hybrid-B' achieves improvements of 6.2% and 11.5% over the 'pipe-only' approach and the 'para-only' approach, respectively.

Our key contributions are summarised as follows:

- We introduce *Para-Pipe*, a partitioning and scheduling framework designed to optimize the inference of modern ML models on heterogeneous SoCs. *Para-Pipe* adeptly balances the trade-offs between latency and throughput

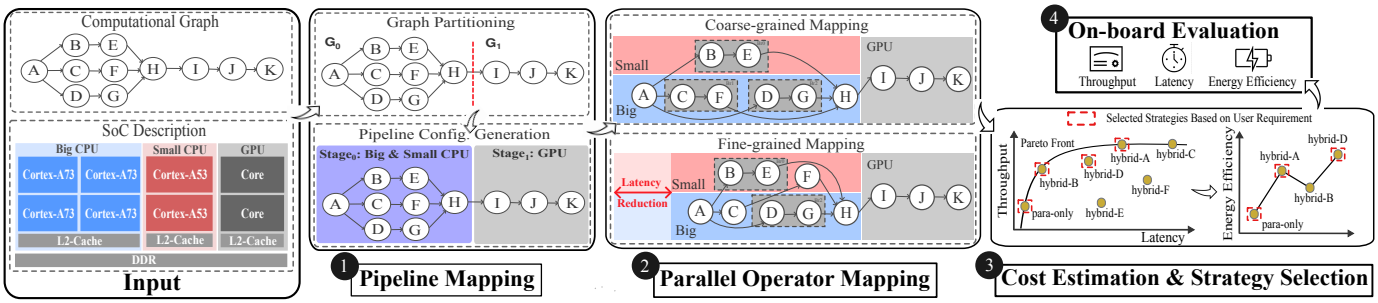


Fig. 3. *Para-Pipe* framework: High-level overview.

while significantly enhancing energy efficiency.

- We propose a two-stage hierarchical mapping approach that effectively leverages operator parallelism within a pipelined structure, facilitating the development of robust strategies for large-scale models.
- We develop two Integer Linear Programming (ILP)-based parallel operator mapping algorithms with distinct granularities, efficiently balancing mapping precision and mapping time.
- We implement *Para-Pipe* runtime with *ARM Compute Library* and evaluate its performance on Amlogic and BST SoC to demonstrate versatility and applicability on real platforms.

II. Overview

Para-Pipe features two levels of mapping: pipeline-level (Section III) and operator-level (Section IV) as illustrated in Fig. 3. In contrast to traditional DAG (Directed Acyclic Graph) mapping algorithms [17]–[19], which are designed for generic task graphs and conventional pipelining techniques [6]–[11], our approach exploits the well-defined structures of most ML computational graphs. This enables us to establish clear boundaries for pipeline stages, optimizing the mapping process more effectively. Given a computational graph sourced from ML frameworks like PyTorch and TensorFlow, along with an SoC specification, Pipeline Mapper identifies subgraphs that exhibit dense operator parallelism and aligns them with a subset of compute engines to form effective pipeline stages. This approach simplifies the graph structure, making it more linear and accessible to partition into pipeline stages, enabling both intra-stage operator and inter-stage pipeline parallelism.

Operator Mapper, tailored for each pipeline stage, assigns operations to compute engines to maximize parallelism and reduce latency. It minimizes costs via ILP. Two algorithms with different granularities are introduced: a coarse-grained algorithm treating a branch as the mapping unit and a fine-grained approach mapping each operator individually. This dual-granularity mapping strategy ensures both generality and efficiency, leveraging multi-branch structures to expedite ILP problem resolution. A cost estimator (Section V-A) is included to model the computational and communication costs required for the ILP formulation.

Para-Pipe evaluates all available strategies using the cost estimator to determine network inference latency, throughput, and energy efficiency. It selects mapping strategies exhibiting Pareto optimality (Section V-B), offering users multiple choices aligned with their application requirements. Currently, *Para-Pipe* operates as a one-time static process, with no runtime dynamicity incorporated. The performance of these mappings is subsequently validated on the real SoC platform using the *Para-Pipe* runtime (Section VI-A).

Algorithm 1: Graph Partitioning

Input: Computational graph G, graph inputs $inputs$, graph outputs $outputs$.
Output: Subgraph list $subgraphs$.

$subgraphs \leftarrow \emptyset$; $sinks \leftarrow outputs$;
$G \leftarrow$ ***mergeNodes***(G);
if ***isLinear***(G) **then**
  foreach $node_{merged} \in G$ **do**
    $G_{sub} \leftarrow$ ***createSubgraph***$(node_{merged})$;
    $subgraphs$.add(G_{sub});
  return $subgraphs$;
while $sinks \neq \emptyset$ **do**
  $sink \leftarrow sinks$.pop();
  $N_{go} \leftarrow \{sink\}$; $N_{stop}, sink_{next}, G_{sub} \leftarrow \emptyset$;
  while $sink_{next} = \emptyset$ ***and*** $N_{go} \neq \emptyset$ **do**
    $B_{end} \leftarrow \emptyset$;
    foreach $node_{go} \in N_{go}$ **do**
      $B_{end} \leftarrow$ ***traceToBranchEnd***$(node_{go}, G_{sub})$;
    $B_{end} \leftarrow B_{end} \cup N_{stop}$;
    $sink_{next} \leftarrow$ ***isTerminalNode***$(B_{end}, inputs)$;
    $N_{stop}, N_{go} \leftarrow$ ***isExtendable***(B_{end});
  end while
  if $sink_{next} = \emptyset$ **then**
    $sinks$.add$(sink)$;
  else
    $sinks$.add$(sink_{next})$; $subgraphs$.add(G_{sub});
end while

III. Pipeline Mapping

A. Graph Partitioning

Modern neural networks often exhibit irregular structures characterized by extensive operator volumes and complex interconnections. To manage this complexity, we partition the computational graph G into non-overlapping, topologically ordered subgraphs, denoted as $G_1, G_2, ...$ Each subgraph exhibits either a linear chain of operator nodes or a fan-in structure. A fan-in structure is characterized by the presence of a single sink node to which all source nodes within the subgraph ultimately connect. These source nodes are either graph inputs or share a common parent node, which serves as the sink node for the preceding subgraph. Fan-in structures

are prevalent in contemporary neural networks, such as the InceptionNet family and transformer-based models, where they facilitate processing image features and sentence structures by consolidating common inputs into a unified output.

Algorithm 1 elaborates the graph partitioning approach. To simplify the computational graph, operator nodes are initially merged wherever feasible to reduce structural complexity. For purely linear networks, the algorithm partitions the graph by treating each operator node as an independent subgraph, thereby broadening the framework's applicability *(Line 3-7)*. For graphs with irregular structures, the algorithm commences by traversing from graph outputs to discern subgraphs adhering to fan-in structures. The graph outputs serve as initial sink nodes, recorded in the set $sinks$. The entire graph is iteratively explored until $sinks$ is empty.

To identify subgraphs, two sets, N_{go} and N_{stop}, are initialized to track extensible and non-extensible nodes, respectively. G_{sub} maintains nodes traced in the current subgraph. Initially, only the sink node is placed in N_{go}, and tracing operations are executed on all nodes within N_{go}.

During tracing, the furthest nodes in the same linear chain as nodes in N_{go} are identified and stored in B_{end} *(Line 14)*. These nodes serve as potential sources. Subsequently, B_{end} is updated with nodes from N_{stop}. If nodes in B_{end} represent inputs or share a common parent *(Line 16)*, the algorithm designates the parent as the sink node for the next subgraph and generates the subgraph G_{sub} *(Line 22)*. Otherwise, N_{go} and N_{stop} are updated based on the extensibility of nodes in B_{end}: for extensible nodes, this update occurs after all their child nodes have been visited, while for non-extensible nodes, the update happens when at least one child node remains unvisited *(Line 17)*. Tracing persists if extensible nodes exist until termination. If no extensible nodes are found, and the traced nodes cannot form a fan-in structure, the sink node is appended to $sinks$ for later consideration *(Line 20)*. Ultimately, acquired subgraphs are ordered based on their topological sequence.

B. Stage Configuration Generation

Consider an SoC with m compute engines, *Para-Pipe* offers pipeline configurations ranging from 1 (para-only) to m (pipe-only) stages. It enumerates processor combinations for each configuration, matching the number of stages while preventing processor overlap and ensuring optimal utilization. Specifically, in each configuration, processors are divided into distinct sets corresponding to each stage. This division guarantees that each processor is uniquely assigned to one set, ensuring a non-overlapping and exclusive allocation of processors across different stages.

Once pipeline stages are linked with specific processor combinations, *Para-Pipe* enumerates potential partition points to generate multiple candidate pipeline configurations, determining the set of sequential subgraphs for each stage. The limited number of on-chip processors and the efficient process of identifying subgraphs ensure minimal time overhead for this enumeration. Notably, this enumeration approach can produce a pipe-only configuration that achieves nearly maximal throughput. Here, a stage is denoted as $(G_{l,k}, P)$, where $l \leq k$, with $G_{l,k}$ representing a graph composed of subgraphs $G_l, G_{l+1}, ..., G_k$ executed on the processor set P. In the case of the Amlogic SoC example in Fig. 3, the initial stage employs two CPU clusters for operator parallelism within the first subgraph, followed by the processing of the second subgraph by a GPU in the subsequent stage. A Parallel Operator Mapping pass is subsequently applied to devise a parallel mapping strategy for the first subgraph.

IV. Parallel Operator Mapping

We introduce two ILP-based algorithms operating at distinct granularity. The coarse-grained method, suitable for graphs with independent branch structures, provides solutions less susceptible to execution jitters and synchronization overhead very fast. Conversely, the fine-grained approach, unconstrained by graph structure, yields superior mapping results but with longer solving times. For example, as depicted in Fig. 3, the fine-grained mapping approach, in contrast to the coarse-grained method, allows for the segmentation of branches and the distribution of operators across CPU clusters, effectively minimizing processor idle time and thus reducing latency.

A. Coarse-grained Mapping

This mapper allocates operators within a single branch to the same processor to minimize potential communication overhead. Given a stage configuration $(G_{l,k}, P)$, we employ it to generate mapping strategies for subgraphs $G_l, G_{l+1}, ..., G_k$ sequentially. Our objective for each subgraph is to minimize the execution time of the processor with the longest duration.

We begin by extracting independent branches from the subgraph into set S. For each branch s, we create a binary variable x_s in Equation (1) to represent its placement strategy. This variable is constrained by $\sum_{p \in P} x_{sp} = 1$, ensuring that a branch is assigned to only one processor. This constraint allows for multiple branches to be allocated to one processor, optimizing the utilization of high-performance processors.

$$x_{sp} = \begin{cases} 1, & \text{if computationally demanding path of} \\ & \text{branch } s \text{ is assigned to processor } p \\ 0, & \text{otherwise.} \end{cases}, \quad (1)$$

In each branch s, we quantify the execution time of the computationally intensive path on processor p as cpc_{sp}, representing the cumulative sum of execution times for all included operators. The sole sink node in the subgraph is allocated to the highest-performance processor q. The communication cost between s and the sink node t is denoted as $cmc_{sp} = cmc_{(s_l,t),pq}$, where s_l represents the tail node of the computationally intensive path in branch s. The execution cost of s on processor p can be represented as $exec_{sp} = cpc_{sp} + cmc_{sp}$.

The longest execution time duration L is formulated as Equation (2). To minimize the value of L, we utilize ILP techniques to determine optimal values of binary variables x, which generates an effective computation path placement for all branches.

$$L = \max_{p \in P} \sum_{s \in S} exec_{sp} \times x_{sp} \quad (2)$$

Our algorithm employs idle or unused processors to address the allocation of operators in nested branches that are not part of the computationally demanding path. It replicates the mapping process to effectively assign these nodes within the nested structures to the appropriate processors, ensuring optimal resource utilization.

B. Fine-grained Mapping

We first introduce the objective function and the execution time modeling, including computational and communication costs. Then, we describe the constraints to formulate the parallelism limited by the availability of processors. In a graph, the objective function is to minimize the maximum execution time of all last operators. An operator v has P possible mapping processors, which is formulated as a vector of length P, each element is a binary variable denoting if v is assigned to processor p, described in Equation (3). x_v is constrained by $\sum_{p=1}^{P} x_{vp} = 1$, which requires an operator to be assigned to only one processor.

$$x_{vp} = \begin{cases} 1, & \text{if operator } v \text{ is assigned to processor } p, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The execution time of v is formulated as Equation (4). st_v refers to the start time of operator v. Operator v starts executing after all predecessors are finished. $exec\_cpc_v$ refers to the actual time spent on computation, which is computed as $\sum_{p=1}^{P} x_{vp} \times cpc_{vp}$, and $exec\_cmc_{(v,u)}$ is the communication cost for edge (v, u) calculated by $\sum_{p=1}^{P}\sum_{q=1}^{P} y_{(v,u),pq} \times cmc_{(v,u),pq}$. cpc and cmc are costs profiled offline and estimated with our cost model described in Section V-A.

$$st_v \geq st_{pred} + exec\_cpc_{pred} + exec\_cmc_{(pred,v)} \tag{4}$$

$y_{(v,u),pq}$ are binary variables representing whether a pair of processors is selected, as shown in Equation (5). They are under similar constraints to x that force only one pair of processors to be selected, $\sum_{p=1}^{P}\sum_{q=1}^{P} y_{(v,u),pq} = 1$. By introducing this variable, we can formulate the communication cost of the edge (v, u) with the same approach of $exec\_cpc$. By multiplying the binary variable and the corresponding cost and summing them together, both the computational and communication time of the graph are formulated.

$$y_{(v,u),pq} = \begin{cases} 1, & \text{if } x_{vp} \wedge x_{uq} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Additionally, we formulate the availability of processors to limit the number of operators executing in parallel. This constraint requires that the execution time of two parallel operators assigned to the same processor cannot overlap. For each pair of parallel operators v and u, we formulate the constraints in (6), where ft represents the operator finish time.

$$ft_v \leq st_u \; OR \; ft_u \leq st_v, \text{if } x_v = x_u \tag{6}$$

We use the big M technique to express the condition and transform (6) to (7).

$$\begin{aligned} ft_v - st_u &\leq M \times (2 - x_{vp} - x_{up}) + M \times z_{uvp} \\ ft_u - st_v &\leq M \times (2 - x_{vp} - x_{up}) + M \times (1 - z_{uvp}) \end{aligned} \tag{7}$$

z_{uvp} is a set of binary variables introduced to formulate the exclusive condition between the two formulas. If $z_{uvp} = 1$, the formula above holds true; otherwise, the below holds true. And ft_v is computed with $st_v + cpc_{vp}$. We solve this ILP problem with gurobi [28].

V. Cost Estimation & Strategy Selection

After the Pipeline Mapping Pass—which details all potential pipeline configurations—and the Parallel Operator Mapping Pass—which establishes mapping strategies within these stages—a comprehensive performance and energy estimation is conducted. *Para-Pipe* identifies Pareto-optimal mapping options based on this estimation. Users can then select from these options to find those that best meet their latency, throughput, and energy efficiency preferences.

A. Cost Estimation

1) Performance: Within a specific mapping option, model latency is defined as the cumulative execution time of all stages, while throughput is defined as the reciprocal of the execution time of the longest stage. For the concurrent execution of processors within a stage, we simulate both the co-execution of processors and the associated communication costs. On Amlogic SoC, operator computation times are systematically profiled offline. The costs associated with operator communication are determined based on the time required for data transition and conversion. This includes profiling SDRAM access times for CPUs and GPU, as well as the overhead associated with data conversion tasks like address mapping. These measurements are carried out using benchmarking tools such as TinyMemBench [29] and clpeak [30], which provide precise insights into memory access and data handling efficiencies under various operational scenarios. For BST SoC, computation and communication costs are derived using its operator simulator, as detailed in Section VI-B.

2) Energy Efficiency: Given one mapping strategy, energy efficiency, quantified as inference requests per unit of energy ($frames/joule$), is computed by dividing its throughput by total platform active power. Due to the absence of an energy estimation model provided by BST [31], this study evaluates energy efficiency exclusively on the Amlogic SoC. The total active power is determined as the sum of the estimated power consumption of all processors. It is important to note that memory power is inherently included in the processor power consumption measurements and is therefore not accounted for separately.

We use the model proposed in [32] to estimate processor power consumption, expressed in Equation (8). At the voltage/frequency level of V_f/f, the processor power is the sum of dynamic power $\alpha V_f^2 U_f f$, where α is capacitance, and U_f is processor utilization, and static power βV_f with the leakage current β. α and β are platform-specific constants. To refine these parameters, we measure variations in power consumption at different processor utilizations and apply linear regression.

$$P_f = \alpha V_f^2 U_f f + \beta V_f \tag{8}$$

Power consumption measurements are facilitated using a USB power meter [33]. This USB power meter quantifies the

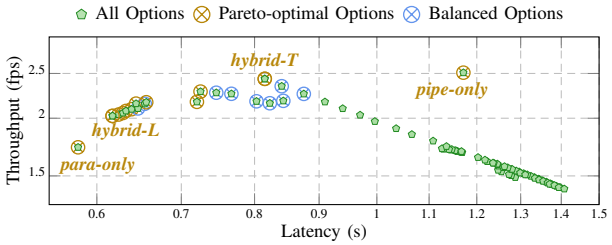


Fig. 4. Pareto optimal analysis of mapping options in *Para-Pipe* for Inception-ResNet-v2 on Amlogic SoC: Balanced options are strategies achieving moderate throughput and latency performance.

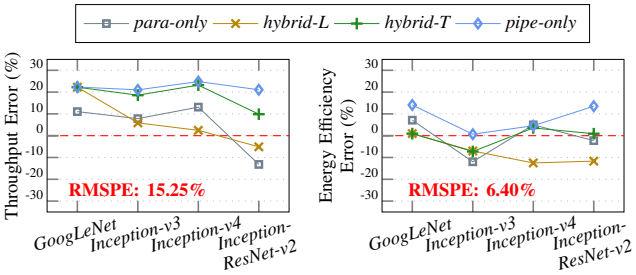


Fig. 5. Prediction error of throughput (left) and energy efficiency (right) of selected *Para-Pipe* mapping options on Amlogic SoC.

power consumption of the entire platform, denoted as P, with all unused processors deactivated to exclude their impact on the total power consumption. To ascertain the influence of non-processor components on the total power, all processors are turned off, and this power measurement is labeled as P_{non}. The power consumption of the specific processor under test is then calculated by subtracting P_{non} from P, represented as $P - P_{non}$.

For GPU utilization, data is sourced from kernel activity logs via sysfs attributes [34]. Sysfs, a RAM-based filesystem, exports kernel data structures, attributes, and linkages to user space, thus facilitating the monitoring of hardware and system metrics. CPU cluster utilization is monitored using the *htop* tool, which helps us calculate the average utilization of included cores, representing the cluster's overall utilization.

After establishing the power estimation equation (8), we measure processor utilization to estimate average power consumption accurately. Since model operators typically do not fully utilize CPU or GPU resources, we directly measure the processor utilization across the entire model. We hypothesize that this measurement effectively represents the utilization of individual operators. Furthermore, by analyzing the processors' spare time indicated by mapping strategies and observing processor utilization during the execution of assigned operators, we can calculate the average processor utilization associated with specific mapping strategies.

B. Strategy Selection

Following the estimation, we offer users a range of Pareto-optimal configuration options tailored to their preferences for throughput and latency trade-offs. Energy efficiency consideration of generated mapping strategies is also included as a selection metric.

1) Pareto Optimal Analysis: Based on estimated latency and throughput, configurations exhibiting inferior performance across both metrics are excluded, while those demonstrating Pareto optimality are retained. Fig. 4 illustrates the performance trade-offs associated with various *Para-Pipe* configurations for Inception-ResNet-v2 on the Amlogic SoC, employing coarse-grained mapping to develop parallel operator strategies.

The pipe-only configuration achieves maximum throughput but suffers from impractical latency levels. Conversely, the para-only setup minimizes latency at the expense of significantly reduced throughput. Hybrid configurations present multiple Pareto-optimal solutions that effectively balance throughput with latency. When both metrics are accorded equal importance (Balanced Options), *Para-Pipe* also yields several commendable configurations for users to choose from. This study focuses on pipe-only, para-only, and hybrid options that achieve the lowest latency and highest throughput (hybrid-L and hybrid-T) for further analysis.

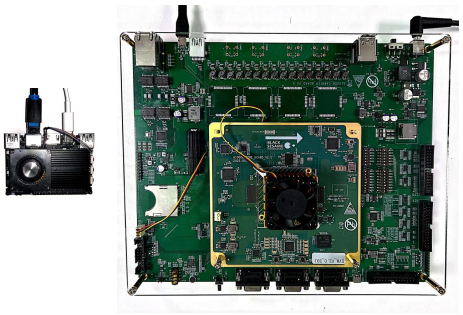


Fig. 6. Picture of Khadas VIM3 Pro board (left) and BST A1000 evaluation board (right).

2) Estimation Accuracy: To assess the effectiveness of our cost estimator, we evaluated it across four configurations: para-only, hybrid-L, hybrid-T, and pipe-only. Our findings indicate that the root mean squared prediction error (RMSPE) for latency, throughput, and energy efficiency stands at 15.33%, 15.25%, and 6.40%, respectively, across all benchmarks. Details of the throughput and energy efficiency prediction errors are illustrated in Fig. 5. Despite an approximate 15% prediction error in latency and throughput, the cost estimator effectively determines the relative performance rankings of mapping strategies for both metrics. These rankings have been verified on the real platform using runtime evaluations, ensuring the reliability of the cost estimator in identifying robust and practical Pareto-optimal solutions.

VI. Runtime Management on SoC

We implement the runtime management of *Para-Pipe* onto *Amlogic A311D SoC of Khadas Vim3 Pro Platform* [35] (Fig. 6). Additionally, we conduct performance estimation using the *BST A1000 evaluation board* (Fig. 9) provided by [31], which is explained in detail in Section VI-B.

A. Implementation on Amlogic SoC

The Amlogic SoC incorporates an *ARM big.LITTLE* CPU architecture, featuring a high-performance quad-core Cortex-A73 cluster and a power-efficient dual-core Cortex-A53 cluster, along with an *ARM* G52 MP4 GPU, as depicted in Fig. 3. Since most ML libraries optimize at the operator level for

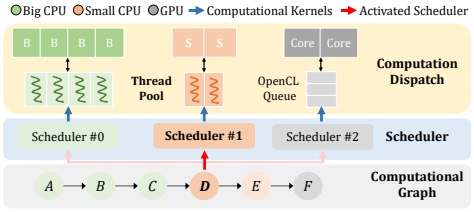
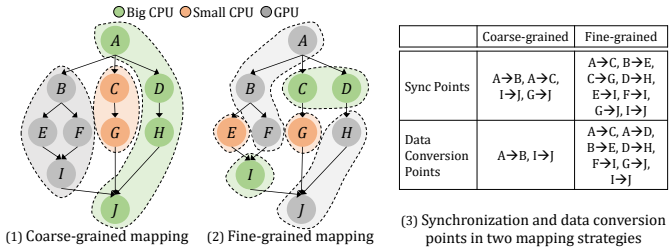


Fig. 7. *ARM-CL* runtime illustration: sequential execution of computational graph across big CPU cluster, small CPU cluster, and GPU on Amlogic SoC.



	Coarse-grained	Fine-grained
Sync Points	A→B, A→C, I→J, G→J	A→C, B→E, C→G, D→H, E→I, F→I, G→J, I→J
Data Conversion Points	A→B, I→J	A→C, A→D, B→E, D→H, F→I, G→J, I→J

(3) Synchronization and data conversion points in two mapping strategies

Fig. 8. Runtime comparison of coarse- vs. fine-grained mapping strategies: Analyzing synchronization and inter-processor communication overhead.

sequential neural networks, *Para-Pipe* can be adapted to any of these libraries to achieve specific latency-throughput trade-offs and improved energy efficiency without compromising operator-level optimization. In this paper, we build the *Para-Pipe* runtime for Amlogic SoC on top of *ARM Compute Library (ARM-CL)*, which provides a collection of state-of-the-art low-level ML functions optimized for *ARM* Cortex-A CPU and Mali GPU architectures. *ARM-CL* features a *Graph* API for constructing a model graph representation employing graph-level optimizations to enhance efficiency. This library employs *Neon* (or *SVE*) for acceleration on *ARM* CPUs and *OpenCL* for GPU.

Since the default *ARM-CL* only supports sequential execution onto CPUs or GPU, we configure unique target backends for CPUs and GPU to distribute graph nodes across processors. Nodes are assigned to specific backends based on our mapping strategy. We employ separate schedulers for CPU clusters and GPU to support the concurrent execution of operators; a simplified illustration of the sequential execution of operators across processors is depicted in Fig. 7. The schedulers for CPUs are associated with a thread pool consisting of worker threads bound to the corresponding CPU cluster cores. During node execution, the scheduler pertinent to the target backend is invoked. The CPU scheduler manages the execution by dispatching computation kernels to worker threads, while the GPU scheduler enqueues computational kernels for asynchronous execution in the *OpenCL* queue.

1) Pipeline Deployment: We generate an *ARM-CL* graph for each pipeline stage and execute it on a single thread. In cases where multiple engines are utilized for computation within a stage, the same number of threads are initiated for independent execution. Meanwhile, a data buffer stores transferred data between consecutive stages. Whenever a stage produces output data, its successor stage is notified.

2) Co-execution of CPUs and GPU: Separate threads are spawned to concurrently execute nodes on both CPU clusters and GPUs, with semaphores employed to manage data dependencies. Regarding inter-node data transfer between CPU clusters, no data conversion is required. However, data conversion is imperative for the intricate communication between CPU clusters and GPU. GPU computation exclusively operates on data formatted in the *OpenCL* tensor format. When GPU computation requires CPU data, a specialized *OpenCL* tensor is created to accommodate the transferred data. Conversely, an additional address mapping is required to facilitate CPU access to GPU output within the device address space. Fig. 8 compares the runtime overhead of coarse- and fine-grained mapping strategies. The comparison indicates that a coarse-grained approach exhibits reduced susceptibility to synchronization overhead and frequent communication demands.

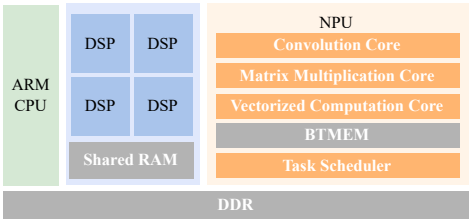


Fig. 9. BST A1000 SoC hardware architectural diagram.

B. Implementation on BST SoC

The BST A1000 SoC offers high-performance, low-power neural network inference. It incorporates an ASIC accelerator NPU and four Cadence DSP cores. It is designed for a wide range of embedded applications, with a central focus on camera-based sensing and AI computing for Level-3 autonomous driving systems. Fig. 9 provides a simplified view of the BST A1000's hardware architecture. In real-world use cases, two DSPs are reserved for other uses in autonomous driving. Therefore, we use a 1 NPU + 2 DSP combination to simulate a realistic scenario.

We utilize an operator simulator provided by [31] to estimate runtimes on both processing engines and the communication times between them. The runtime and communication cost estimation for operators utilizes historical data from previously tested configurations as a benchmark. For operators without prior testing under diverse parameter settings, we apply a k-means clustering algorithm to predict runtime and communication cost based on operator similarities. These estimations serve as the foundation for applying our *Para-Pipe* framework, providing data for the ILP formulation in the Parallel Operator Mapper and enabling the estimation of latency and throughput for the generated mapping strategies.

VII. Experimental Evaluation

This section evaluates the *Para-Pipe* framework on Amlogic and BST SoCs, focusing on selected strategies, including pipe-only, para-only, and hybrid options targeting the lowest latency (hybrid-L) and the highest throughput (hybrid-T). All experimental data for the Amlogic SoC, encompassing latency, throughput, and energy efficiency, were obtained from the real Khadas Vim3 Pro platform. Conversely, the data for the BST SoC are derived from simulations.

TABLE I
STRUCTURAL DETAILS OF BENCHMARK MODELS

Network	#Subgraphs	Maximum Parallelism	Major Operators / Modules
GoogLeNet	11	4	3 Conv + 9 Inception Modules + 1 FC
Inception-v3	13	6	5 Conv + 11 Inception Modules + 1 FC
Inception-v4	21	6	11 Conv + 14 Inception Modules + 2 Reduction Modules + 1 FC
Inception-ResNet-v2	45	4	13 Conv + 40 Residual Inception Modules + 2 Reduction Modules + 1 FC
PETR-based	24	16	ResNet-50 + 3D Position Encoder + Transformer Decoder + Query Generator
BEVFormer-based	25	4	6 Spatial Cross-attention + 6 Transformer + 6 Temporal Self-attention

A. Experimental Setup

1) Benchmarks: The pipelined execution of linear models was comprehensively evaluated in our prior work [11] on the Amlogic SoC. This study focuses on six modern models with dense operator parallelism to evaluate the performance of *Para-Pipe*. The chosen models, GoogLeNet, Inception-v3, Inception-v4 [36], and Inception-ResNet-v2 [36] exemplify conventional regular DNNs with distinct independent branch structures. Two networks featuring complex and irregular connections, i.e., PETR-based network [5] and BEVFormer-based [12], are the state-of-the-art in autonomous driving. Some operators included are not supported by *ARM-CL* on Amlogic SoC; so we evaluate them on BST SoC to demonstrate the platform portability of our framework.Table I outlines model structural details, including the number of subgraphs generated by the Graph Partitioner, serving as partition points in *Para-Pipe* Pipeline Configuration Generator.

2) Platform Setup: We assess the performance and efficiency of the Amlogic SoC using a real Khadas Vim3 Pro platform. In our experiments, the platform's big and small CPU clusters are set to operate at their maximum voltage and frequency levels—1.04 V/2.2 GHz and 0.83 V/1.8 GHz, respectively. Additionally, the *ARM* GPU on the platform is configured to run at its peak performance level of 1.15 V/0.8 GHz. We connect the board to a host machine via an Ethernet cable using Secure Shell (SSH) for reliable and secure data transfer. A Khadas-provided DC 5 V cooling fan is utilized throughout testing to mitigate potential thermal instabilities.

We input a continuous stream of 50 frames for each mapping strategy and measure the average throughput, quantified as frames processed per second and the average latency per frame. A USB power meter [33] is employed to monitor the total power consumption of the board, from which we calculate the average active power and energy efficiency. Following each run, the board is left idle to cool down adequately before proceeding with the next test.

3) Baseline: In this study, we establish two baseline configurations. Firstly, we employ the Layer-switched algorithm [37] to illustrate the performance capabilities of traditional sequential execution of models utilizing all on-chip processor resources. The Layer-switched algorithm supports layer-wise transitions among processors and follows a sequential execution strategy to minimize latency by effectively managing the switching overhead between processors. Notably, the algorithm achieves superior latency and throughput performance without leveraging operator or pipeline parallelism, thus serving as an upper bound for sequential execution performance when utilizing any single or multiple hardware platform components. This evaluation on Amlogic SoC utilizes the tool reported in [9].

TABLE II
SUBGRAPH ALLOCATIONS FOR HYBRID-L AND HYBRID-T STRATEGIES ACROSS FOUR REPRESENTATIVE MODELS IN *Para-Pipe* ON BST SOC

Network	Option	Pipeline Config.	Subgraph Allocation
GoogLeNet	hybrid-L	NPU - 2DSPs	[1,10] - [11]
	hybrid-T	NPU - 2DSPs	[1,9] - [10,11]
Inception-v4	hybrid-L	NPU+DSP - DSP	[1,20] - [21]
	hybrid-T	DSP - NPU+DSP	[1,3] - [4,21]
PETR-based	hybrid-L	NPU+DSP - DSP	[1,17] - [18,24]
	hybrid-T	NPU+DSP - DSP	[1,15] - [16,24]
BEVFormer-based	hybrid-L	NPU+DSP - DSP	[1,23] - [24,25]
	hybrid-T	DSP - NPU+DSP	[1,2] - [3,25]

Secondly, we implement and compare two classic DAG mapping algorithms, HEFT and CPOP, as detailed in [19]. For each tested model, we select the algorithm that demonstrates optimal latency to serve as a comparative baseline in our experiments, and we refer to this algorithm as HEFT & CPOP throughout this paper. HEFT algorithm prioritizes tasks based on the highest upward rank value at each decision point, assigning tasks to the processor that enables the earliest completion. In contrast, CPOP integrates both upward and downward ranks to prioritize tasks, specifically scheduling those on the critical path to the processor that minimizes total execution time. We use these algorithms to evaluate our parallel operator mapping algorithms rigorously.

B. Resultant Configurations

This section briefly analyzes the pipelined configurations of the hybrid-L and hybrid-T options in *Para-Pipe*. On Amlogic SoC, both options utilize a dual-stage configuration employing CPU clusters and the GPU across all supported models. The configuration utilizes a coarse-grained algorithm for parallel operator mapping within the stages. This is because parallel processing within a single stage prefers processors with similar architectures and data formats to reduce unnecessary data conversion and communication overhead. The details are further explained in Section VII-F. Conversely, the partial operator support of BST SoC's NPU, necessitating the offloading of certain operators to DSP, limits the feasibility of *Para-Pipe*'s ideal operator or stage arrangement. Therefore, for most benchmark models, the hybrid options mostly combine NPU and DSP in one stage while solely using DSP in another. The DSP is leveraged not only as a fallback for operators unsupported by the NPU but also to exploit operator parallelism within subgraphs when feasible. The irregular models evaluated on BST SoC necessitate a fine-grained mapping approach.

To understand how *Para-Pipe* selects pipeline structures with preferences for latency and throughput, we focus on the BST SoC and select four representative benchmarks for enhanced visualization. The allocations for the hybrid-L and hybrid-T strategies are detailed in Table II. The hybrid-T option typically maintains a balanced pipeline configuration, as it enhances throughput by decreasing delays in the longest stage while concurrently distributing a significant workload to other

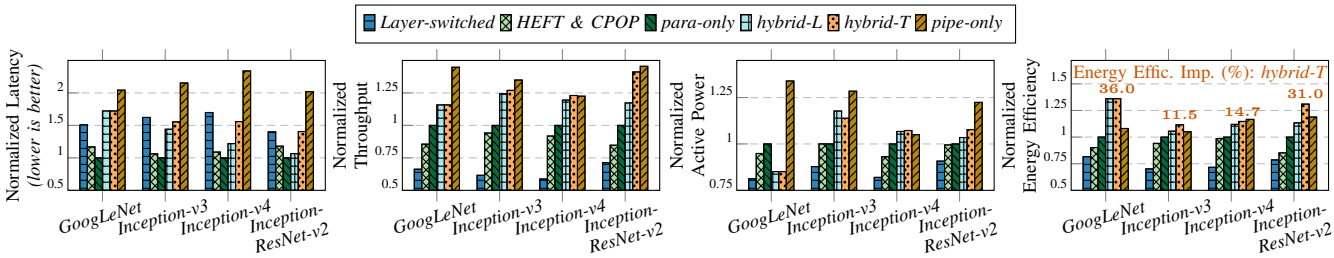


Fig. 10. Normalized latency, throughput, active platform power, and energy efficiency for Layer-switched method, HEFT & CPOP algorithm and four representative *Para-Pipe* mapping strategies on Amlogic SoC relative to the para-only option.

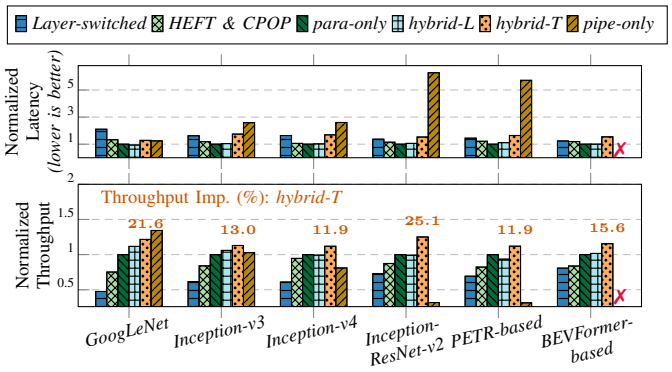


Fig. 11. Normalized latency and throughput for Layer-switched method, HEFT & CPOP algorithm and four representative *Para-Pipe* mapping strategies on BST SoC relative to the para-only option.

stages. Based on the data in Table II, we identify two methods by which the hybrid-L option accelerates execution per frame relative to the hybrid-T option. The first method maps more subgraphs onto the high-performance NPU, decreasing the overall execution time for a single frame, as demonstrated in the GoogLeNet configuration. The second method boosts intra-stage operator parallelism by allocating additional subgraphs for parallel processing, as seen in the configurations for remaining networks. *Para-Pipe* can adaptively adjust workloads within and across stages to optimize latency and throughput trade-offs through hierarchical mapping passes.

C. Latency-Throughput Trade-off

We normalize these metrics relative to the para-only method to explore the balance between latency and throughput. The comparative results for Amlogic SoC are depicted in Fig. 10 and for BST SoC in Fig. 11.

Our findings indicate that traditional sequential execution modes fail to fully utilize on-chip computing resources even with layer-wise transitions among processors when models exhibit intensive operator parallelism. The Layer-switched algorithm, although effective, cannot surpass the performance of purely parallel operator execution.

Our parallel operator mapping algorithm (para-only) significantly improves over the better-performing baseline algorithm, either HEFT or CPOP. Specifically, it achieves an average latency improvement of 10.9% on Amlogic SoC and 15.5% on BST SoC, and throughput improvements of 12.5% and 18.8%, respectively. Accurate profiling of operator runtime statistics, combined with ILP-based techniques for solving parallel mapping problems, empowers our algorithm to devise strategies that optimize for minimal execution time.

However, purely parallel operator execution does not necessarily yield higher throughput than pipelined execution, while pipelined execution inevitably causes a higher overhead in latency. For example, on Amlogic SoC, the pipe-only configuration approaches the highest throughput but results in an average of 113.8% increase in latency compared to the para-only setup. Conversely, the para-only setup achieves the shortest execution times, albeit with an average of 26.6% reduction in throughput. Hybrid strategies that prioritize either throughput or latency offer a viable compromise. The hybrid-L configuration, for instance, decreases throughput by 12.4% but significantly reduces latency by 36.0% compared to the pipe-only approach. Similarly, the hybrid-T strategy decreases throughput by 7.3% but improves latency by 26.8%. In the case of Inception-v4, the hybrid-T strategy excels, surpassing the pipe-only configuration in throughput while halving the latency.

D. General Applicability

Graph-aware modulation of intra- and inter-stage operator parallelism enables *Para-Pipe* to effectively navigate platform constraints, such as limited processor support for certain operators, by offering various mapping strategies. Here, we utilize BST SoC to illustrate this general applicability of our framework *Para-Pipe*. As shown in Fig. 11, limited processor support significantly undermines the efficacy of pipe-only optimizations in throughput. Specifically, unsupported operators may be situated at crucial partition points, rendering an ideal pipe-only configuration infeasible. Compared to the para-only strategy, the pipe-only option results in substantial latency overheads without providing throughput benefits for most models. For instance, in the PETR-based model, the pipe-only configuration incurs a latency that is 5.7 times greater and a throughput that is only 31.4% of that achieved with the para-only method. Moreover, generating an efficient pipe-only configuration for a BEVFormer-based network is infeasible. Conversely, the hybrid-T option, which utilizes one DSP to support operators that the NPU alone cannot, during the parallel processing stage, achieves an average throughput improvement of 16.2% across all tested models relative to the para-only strategy. This represents the highest throughput enhancement across the majority of tested models. Regarding latency, the hybrid-L and para-only options perform comparably. These results verify that *Para-Pipe* can successfully

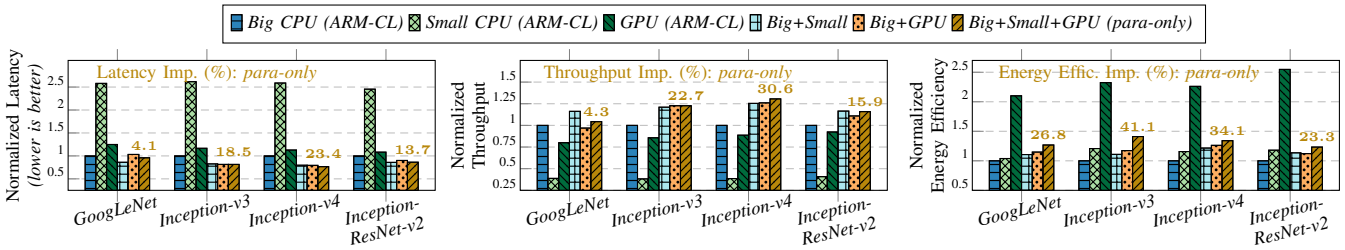


Fig. 12. Normalized latency, throughput, and energy efficiency for coarse-grained non-pipelined, parallel co-execution on Amlogic SoC relative to sequential execution on its big CPU cluster.

recommend adaptive mapping strategies, considering platform compatibility, to align with users' preferences.

E. Energy Efficiency

This section uses Amlogic SoC to evaluate the energy efficiency of *Para-Pipe*. Fig. 10 showcases the normalized average active platform power and energy efficiency relative to the para-only strategy. Our analysis reveals that the Layer-switched and HEFT & CPOP algorithms exhibit average degradations of 24.5% and 8.0% in energy efficiency, respectively, compared to para-only. These declines are primarily attributed to frequent processor switching, a scheduling bias toward high-performance but less energy-efficient components, and additional communication overhead due to suboptimal parallel operator mapping.

Compared to HEFT & CPOP, para-only employs optimized workload mapping strategies, which reduce synchronization contention and processor idle time. Consequently, its additional area overhead primarily results from increased runtime processor utilization rather than inefficient hardware resource consumption. Runtime measurements, averaged across tested models, indicate a modest 3.3% increase in platform active power, improved memory access patterns, and an 11.7% reduction in mutex contention. These trade-offs are well-justified by the substantial performance and energy efficiency benefits, making para-only highly suitable for workloads demanding low latency and high energy efficiency.

Conversely, the pipe-only configuration consistently outperforms the para-only setup across all tested benchmarks, achieving an average improvement of 12.2%. Additionally, the hybrid-L and hybrid-T configurations demonstrate average increases in energy efficiency of 16.7% and 23.3%, respectively, relative to the para-only approach. When compared to the pipe-only configuration, these gains are 8.7% for hybrid-L and 11.0% for hybrid-T, underscoring their enhanced performance.

An exception is observed with Inception-v4, where the hybrid-T option reaches 0.53 fps/J, slightly underperforming against the pipe-only's 0.54 fps/J, translating to a minor setback of 1.7%. This slight decrease in energy efficiency with Inception-v4 can be ascribed to a marginal 0.4% increase in throughput and a 33.3% improvement in latency offered by the hybrid-T option. To encapsulate, the hybrid-T option enhances energy efficiency and presents a nuanced trade-off in performance, particularly noticeable in models with extensive operator parallelism.

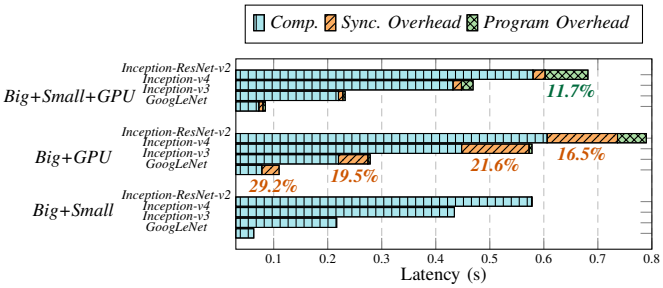


Fig. 13. Latency distribution of fine-grained mapping for non-pipelined, parallel co-execution on Amlogic SoC: computation, synchronization overhead, and program overhead.

F. Co-execution of Heterogeneous Units

This section evaluates the performance of coarse- and fine-grained mapping algorithms in non-pipelined parallel execution via various on-chip processor combinations to demonstrate the efficacy of the operator mapping approaches. We implement and test the strategies generated to ensure a robust analysis, collecting corresponding experimental data on an Amlogic SoC, specifically using a real Khadas Vim3 Pro platform.

1) Coarse-grained Mapping: Fig. 12 illustrates performance and energy gains achieved through coarse-grained co-execution mapping strategies: CPU clusters, big CPU cluster with GPU, and CPU clusters with GPU, relative to the highest-performance big CPU cluster. Parallel processing with CPU clusters obtains an average latency decrease of approximately 16.4% and a throughput increase of about 19.8%. However, the setup with a big CPU and a GPU only marginally outperforms the dual CPU cluster configuration, even if the small CPU cannot deliver even half the performance of the GPU. For models exhibiting lower degrees of parallelism, such as GoogLeNet and Inception-ResNet-v2, the advantages of CPU-GPU co-execution were offset by communication overheads, making it less beneficial. Despite utilizing two CPU clusters and a GPU for parallel inference, the improvement was relatively limited compared to the dual CPU cluster option. Thus, employing purely operator-based parallel execution using heterogeneous on-chip computing resources is not advisable except where extreme latency reduction is required.

Regarding energy efficiency, since the small CPU and GPU are specifically optimized for power efficiency, integrating a big CPU with either a small CPU, GPU, or both can significantly boost energy efficiency. The corresponding combinations deliver average energy efficiency gains of 14.2%, 17.6%, and 31.3%, respectively. These improvements primarily stem

from reduced reliance on the power-intensive big CPU cluster and reduced execution times by engaging more processors. However, while the GPU is recognized for its exceptional power efficiency, its combined use with CPUs somewhat dilutes these benefits.

Consequently, the pipe-only and various hybrid configurations in *Para-Pipe*, which integrate sequential processing stages featuring the GPU, more effectively preserve the GPU's contribution to power efficiency. Besides, selecting processors with compatible computing architectures and data formats, like big and small CPU clusters, for parallel processing within stages is more advantageous. These strategies enhance throughput with minimal latency trade-offs and reduce unnecessary communication and data conversion overhead, thus improving the efficient and effective utilization of on-chip resources.

2) Fine-grained Mapping:

a) Overhead Analysis: We assess latency distributions for fine-grained mapping strategies by comparing them to those obtained from coarse-grained mappings across both runtimes. Program overhead, defined as the additional runtime introduced by the implementation complexity of fine-grained mapping, is estimated as the difference between coarse-grained mapping performance on coarse- and fine-grained runtimes. The computational cost associated with fine-grained mapping is reflected in the lower execution times observed for both strategies. Synchronization overhead is calculated as the difference between the execution time of a fine-grained strategy and the sum of its program overhead and computational cost.

As illustrated in Fig. 13, fine-grained mapping incurs an average program overhead of 5.9% during the concurrent execution of CPU clusters and GPUs, notably impacting large-scale model inferences, such as the 11.7% overhead observed in Inception-ResNet-v2. Synchronization overhead averages 21.7% across tested models when using a large CPU cluster and GPU but decreases significantly to 4.7% with the addition of a small CPU cluster. The inclusion of the small CPU cluster redistributes operators more evenly, alleviating GPU workloads and reducing CPU-GPU synchronization points by 20.1% on average. This, in turn, minimizes CPU-GPU communication and streamlines coordination. Conversely, configurations with only a large CPU cluster and GPU experience frequent and complex CPU-GPU communications. These are further exacerbated by the asynchronous and intermittent command submissions to the *OpenCL* queue, resulting in execution jitters that disrupt synchronization efficiency.

Despite these challenges, fine-grained mapping is markedly more effective for aligning irregular model structures with units that share similar architectures and data formats, as demonstrated with two CPU clusters in our study. In scenarios involving the co-execution of two CPU clusters, the fine-grained approach circumvents the complexities of synchronization and program implementation, yielding an average improvement of 3.35% in latency and 4.28% in throughput compared to coarse-grained mapping.

b) Operator Mapping Choice: The coarse-grained method rapidly generates parallel operator mapping strategies for graphs with independent branches, minimizing processor switching and complex synchronization requirements. This approach is remarkably robust against execution jitters and runtime interruptions from the operating system, making it well-suited for more volatile environments. Conversely, when applied to these relatively straightforward graphs, the fine-grained method performs best in stable runtime environments with uniform computing architectures. Moreover, it excels in mapping complex and irregular model graphs with solution times that scale appropriately with the degrees of operator parallelism.

TABLE III
ILP SOLUTION TIME (MINUTES) FOR FINE-GRAINED MAPPING ALGORITHM FOR NON-PIPELINED, PARALLEL EXECUTION WITH NPU AND TWO DSPS (*para-only*) ON BST SOC: PERFORMANCE EVALUATION ON INTEL(R) XEON(R) GOLD 6326 CPU @ 2.90GHZ

Network	#Subgraphs	#Operators	Time (minutes)
GoogLeNet	11	141	< 1
Inception-v3	13	220	4
Inception-v4	21	343	4
Inception-ResNet-v2	45	576	<1
PETR-based	24	337	361
BEVFormer-based	25	257	<1

G. Mapping time Analysis

The dual parallel mapping methods of *Para-Pipe* enhance the parallel scheduling of subgraphs by generating multiple ILP problems, thereby increasing the efficiency of scheduling large models. Due to its detailed granularity, fine-grained mapping inherently demands longer resolution times than coarse-grained mapping. Consequently, this discussion concentrates on fine-grained mapping to comprehensively assess the scheduling time overhead associated with our framework.

Table III presents solution times for the fine-grained algorithm when generating the para-only option on the BST SoC. Once parallel operator mapping strategies are established for each subgraph, no further ILP computations are necessary. *Para-Pipe* utilizes these results to efficiently generate additional mapping options, such as hybrid-T and hybrid-L, with minimal overhead. Thus, ILP times shown in Table III approximate the total scheduling time required to generate all Pareto-optimal points.

By constructing subgraph-based ILP problems, we can rapidly devise efficient mapping solutions for most networks within 5 minutes. For the PETR-based network, the largest subgraph contains 169 operators and exhibits an inter-operator parallelism degree of 16, making it the most complex subgraph analyzed in this study (its partial architecture with 67 operators is detailed in Fig. 1). The ILP solution time for this subgraph is approximately 6 hours. For large-scale models with multiple complex subgraphs, such as those found in PETR-based networks, subgraph mappings can be scheduled in parallel to expedite the solution process significantly. Unlike meta-heuristic or heuristic algorithms, our subgraph-based ILP approach swiftly generates optimal results for each subgraph without requiring time-consuming iterative adjustments. This approach ensures the optimal solution is found within a tolerable timeframe, even for highly intricate subgraphs.

VIII. Related Work

Optimizing ML inference on heterogeneous SoCs has garnered extensive research interest. Brakel et al. [38] have explored model parallelism within distributed infrastructures, focusing on intra- and inter-operator parallelism, and highlighted their findings through case studies on multi-billion parameter transformer models. This paper aims to explore ML inference on SoCs further, concentrating on the following key dimensions:

a) Pipelining: Pipe-it [6] exploits inter-operator parallelism through pipelining to engage *ARM big* and *LITTLE* CPU clusters concurrently, enhancing throughput. Similarly, PipeBERT [39] focuses on pipelining BERT models onto heterogeneous CPU clusters, employing an improved binary search algorithm to optimize pipeline configurations. Mukherjee et al. [40] and Flexi-BOPI [10] utilize dynamic programming and Bayesian optimization respectively to automate the partitioning process. While these strategies significantly enhance throughput, they do not always ensure acceptable execution latency, especially when applied to large-scale modern neural networks. This limitation poses challenges for applications where low latency is critical.

b) Operator Parallel Execution: Tumeo et al. [17] introduce an *Ant Colony Optimization* algorithm for mapping tasks onto heterogeneous multiprocessor architectures. A Mixed ILP model is developed in [18] to optimize task mapping by considering the data exchange overhead among processors. Topcuoglu et al. [19] propose two classic DAG mapping algorithms, utilizing defined upward and downward rank values to prioritize task scheduling. Zhang et al. [20] devise reliable scheduling for parallel task execution in heterogeneous computing systems, emphasizing both task reliability and energy efficiency. These methodologies are primarily designed for handling task graphs with limited operator parallelism and assume the presence of random and irregular structures, which do not always align with the structured nature of many real-world applications. Additionally, DUET [16] partitions complex-structured DNNs into subgraphs and employs a greedy-correction subgraph scheduling algorithm for near-optimal placement across coupled CPU-GPU architectures. This approach primarily targets multi-model tasks without delving into fine-grained operator-level parallelism.

c) Intra- and Inter-Operator Parallelism: Minakova et al. [8] generate the execution pipeline using CPUs at the operator-level granularity, with GPUs accelerating heavy computation within operators. Similarly, Jeong et al. [41] highlight parallelization using GPU and NPU via pipelining and multithreading on TensorRT. *Synergy* [42] proposes a dynamic runtime strategy, work-stealing, to adjust workload mapping configuration on SoCs. Despite these advancements, the focus remains predominantly on linear models, leaving room for optimization in more complex and irregular model architectures.

d) Multiple Workloads: Kim et al. [43] present an energy-aware pipeline-based mapping methodology for multiple DL applications onto multi-processors. They focus on the dynamic scheduling of applications to reduce energy consumption via DVFS under latency constraints, with each application being a simple and linear graph. Moreover, DAIMO-NPU [44] adopts a cyclic scheduling algorithm to support dynamic addition and removal of model execution tasks onto heterogeneous NPU settings. BAND [45] orchestrates multi-DNN workloads on heterogeneous compute engines, partitioning models into subgraphs and dynamically scheduling these subgraphs for optimized execution. These strategies are crafted for scenarios involving the scheduling of multiple applications under latency constraints without extensively exploring the intricate model structures characteristic of single, complex, and computation-intensive application inferences.

TABLE IV
Comparing *Para-Pipe* with other frameworks that support ML inference on heterogeneous SoCs

Framework	Irregular Model Graph[a]	Pipelineing	Operator Parallel Execution	Strategy Selection[b]
[16]–[20]	✗	✗	✓	✗
[6]–[11], [39], [40]	✗	✓	✗	✗
Para-Pipe	✓	✓	✓	✓

[a] A complex neural network with intensive operator parallelism and interdependencies, unlike simpler linear networks like MobileNet and ResNet50. [b] Selection of mapping strategies based on prioritized metrics like latency or energy efficiency.

Our work innovatively addresses the processing of irregular modern model graphs by leveraging hierarchical operator parallelism within a pipelined architecture. This meticulous tuning of inter- and intra-stage operator parallelism significantly broadens the scope of ML inference optimization on heterogeneous SoCs. Our method achieves a balanced trade-off between latency and throughput and boosts the platform's energy efficiency. Additionally, it generates multiple Pareto-optimal strategies, accommodating diverse user preferences across various performance metrics. Table IV underscores the critical distinctions between our methodology and existing works, highlighting our contributions to the field.

IX. Conclusion

This paper presents *Para-Pipe*, a novel partitioning and scheduling framework that optimizes the trade-offs between latency and throughput to maximize energy efficiency on embedded computing platforms. *Para-Pipe* efficiently maps modern neural networks across heterogeneous processors by leveraging hierarchical operator parallelism within computational graphs. The proposed methodology decomposes the model graph into subgraphs with varying degrees of operator parallelism and assigns them to different computing units for parallel or sequential execution in distinct pipeline stages. This adaptive fusion of parallel and sequential execution enables fine-grained control over latency-throughput trade-offs, enhancing overall energy efficiency. Experimental evaluations on two distinct SoCs (Amlogic and BST) demonstrate *Para-Pipe*'s ability to generate Pareto-optimal configurations, balancing latency and throughput while identifying the most energy-efficient mapping strategies for networks with dense inter-operator parallelism.

References

[1] T. Mitra, "Heterogeneous multi-core architectures," *Information and Media Technologies*, vol. 10, no. 3, pp. 383–394, 2015.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[3] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 578–594.

[4] ARM, "Compute library: A software library for machine learning," 2023. [Online]. Available: https://www.arm.com/technologies/compute-library

[5] Y. Liu, T. Wang, X. Zhang, and J. Sun, "PETR: Position embedding transformation for multi-view 3D object detection," in *European Conference on Computer Vision*. Springer, 2022, pp. 531–548.

[6] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput CNN inference on embedded ARM big. LITTLE multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2254–2267, 2019.

[7] H.-I. Wu, D.-Y. Guo, H.-H. Chin, and R.-S. Tsay, "A pipeline-based scheduler for optimizing latency of convolution neural network inference over heterogeneous multicore systems," in *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 46–49.

[8] S. Minakova, E. Tang, and T. Stefanov, "Combining task-and data-level parallelism for high-throughput CNN inference on embedded CPUs-GPUs MPSoCs," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS, Samos, Greece, July 5–9, Proceedings 20*. Springer, 2020, pp. 18–35.

[9] E. Aghapour, D. Sapra, A. Pimentel, and A. Pathania, "ARM-CO-UP: ARM COoperative Utilization of Processors," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[10] Z. Wang, P. Yang, B. Zhang, L. Hu, W. Lv, C. Lin, and Q. Wang, "Flexi-BOPI: Flexible granularity pipeline inference with Bayesian optimization for deep learning models on HMPSoC," *Information Sciences*, p. 120984, 2024.

[11] E. Aghapour, Y. Zhang, A. Pathania, and T. Mitra, "Pipelined CNN inference on heterogeneous multi-processor system-on-chip," in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Software Optimizations and Hardware/Software Codesign*. Springer, 2023, pp. 405–427.

[12] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao, and J. Dai, "BEVFormer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers," in *European conference on computer vision*. Springer, 2022, pp. 1–18.

[13] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, 2018.

[14] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, "A review of machine learning and IoT in smart transportation," *Future Internet*, vol. 11, no. 4, p. 94, 2019.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[16] M. Zhang, Z. Hu, and M. Li, "DUET: A compiler-runtime subgraph scheduling approach for tensor programs on a coupled CPU-GPU architecture," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 151–161.

[17] A. Tumeo, C. Pilato, F. Ferrandi, D. Sciuto, and P. L. Lanzi, "Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. IEEE, 2008, pp. 142–149.

[18] A. Bender, "MILP based task mapping for heterogeneous multiprocessor systems," in *Proceedings EURO-DAC'96. European Design Automation Conference with EURO-VHDL'96 and Exhibition*. IEEE, 1996, pp. 190–197.

[19] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[20] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Information Sciences*, vol. 319, pp. 113–131, 2015.

[21] V. Tsakanikas and T. Dagiuklas, "Video surveillance systems-current status and future trends," *Computers & Electrical Engineering*, vol. 70, pp. 736–753, 2018.

[22] L. Cheng, J. Wang, and Y. Li, "Vitrack: Efficient tracking on the edge for commodity video surveillance systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 723–735, 2021.

[23] J. Chen, K. Li, Q. Deng, K. Li, and S. Y. Philip, "Distributed deep learning model for intelligent video surveillance systems with edge computing," *IEEE Transactions on Industrial Informatics*, 2019.

[24] H. Bouzidi, M. Odema, H. Ouarnoughi, S. Niar, and M. A. Al Faruque, "Map-and-Conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous MPSoCs," in *60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[25] X. Chen, A. Krishnakumar, U. Ogras, and C. Chakrabarti, "PED: Probabilistic energy-efficient deadline-aware scheduler for heterogeneous SoCs," *Journal of Systems Architecture*, vol. 147, p. 103051, 2024.

[26] L. Amlogic, "Amlogic AI processor A311D datasheet," 2023. [Online]. Available: https://dl.khadas.com/hardware/VIM3/Datasheet/A311D_Datasheet_01_Wesion.pdf

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[28] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2022.

[29] S. Siamashka, "TinyMemBench," 2019. [Online]. Available: https://github.com/ssvb/tinymembench

[30] K. Bhat, "clpeak," 2015. [Online]. Available: https://github.com/krrishnarraj/clpeak

[31] L. Black Sesame Technologies Co., "Black Sesame Technologies," 2024. [Online]. Available: https://www.blacksesame.com.cn/en/

[32] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[33] L. RDTech, "UM25C," 2023. [Online]. Available: https://sigrok.org/wiki/RDTech_UM_series#UM25C

[34] P. Mochel, "The sysfs filesystem," in *Linux Symposium*, vol. 1. The Linux Foundation San Francisco, CA, USA, 2005, pp. 313–326.

[35] L. Khadas Tech. Co., "Khadas Vim3 Pro," 2023. [Online]. Available: https://www.khadas.com/vim3

[36] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[37] E. Aghapour, D. Sapra, A. Pimentel, and A. Pathania, "CPU-GPU layer-switched low latency CNN inference," in *25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2022, pp. 324–331.

[38] F. Brakel, U. Odyurt, and A.-L. Varbanescu, "Model parallelism on distributed infrastructure: A literature review from theory to llm case-studies," *arXiv preprint arXiv:2403.03699*, 2024.

[39] H.-Y. Chang, S. H. Mozafari, C. Chen, J. J. Clark, B. H. Meyer, and W. J. Gross, "PipeBERT: high-throughput BERT inference for ARM big. LITTLE multi-core processors," *Journal of Signal Processing Systems*, vol. 95, no. 7, pp. 877–894, 2023.

[40] A. Mukherjee and S. Dey, "Automated deep learning model partitioning for heterogeneous edge devices," in *Proceedings of the Second International Conference on AI-ML Systems*, 2022, pp. 1–8.

[41] E. Jeong, J. Kim, S. Tan, J. Lee, and S. Ha, "Deep learning inference parallelization on heterogeneous processors with TensorRT," *IEEE Embedded Systems Letters*, vol. 14, no. 1, pp. 15–18, 2021.

[42] G. Zhong, A. Dubey, C. Tan, and T. Mitra, "Synergy: An HW/SW framework for high throughput CNNs on embedded heterogeneous SoC," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, pp. 1–23, 2019.

[43] J. Kim and S. Ha, "Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints," *IEEE Transactions on Computers*, 2022.

[44] K. Sohn, I. Choi, S. Kim, J. Lee, J. Lee, and J. Kim, "A strategy to maximize the utilization of AI neural processors on an automotive computing platform," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2024, pp. 1–4.

[45] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "BAND: coordinated multi-DNN inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 235–247.

Yujie Zhang received the B.S. degree in software engineering from Shandong University, Jinan, China, in 2020. She is currently pursuing a Ph.D. degree at the National University of Singapore, Singapore.

Her research focuses on the efficient deployment of deep learning models on resource-constrained systems.

Huiying Lan received the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2019. She was a Research Fellow at the National University of Singapore, Singapore, from 2023 to 2024. She is currently a Senior Software Engineer at Lumai, Oxford, UK, leading the design and development of software stacks.

Her research interests include AI compiler, inference engine, and software-hardware co-design.

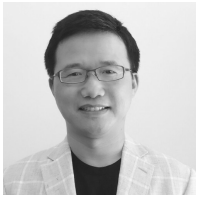
Ehsan Aghapour received the B.S. and M.S. degrees from Sharif University of Technology, Tehran, Iran, in 2016 and 2019, respectively. He is currently pursuing a Ph.D. degree at the University of Amsterdam, Netherlands.

His research interests include Edge AI and low-power system design.

Zhiyuan Ning received the B.S. and M.S. degrees from the University of Southern California, CA, USA, in 2020 and 2022, respectively. He was a Senior AI Framework Software Engineer at Black Sesame Technologies, Inc., CA, USA, from 2022 to 2024. He is currently a Senior AI Framework Software Engineer at Advanced Micro Devices, Inc. (AMD), CA, USA.

Peng Zan received the Ph.D. degree in electrical engineering from the University of Maryland, MD, USA, in 2019. He was a Senior AI Framework Software Engineer at Black Sesame Technologies, Inc., CA, USA, from 2021 to 2023. He is currently a Staff Engineer in AI Software at Samsung Semiconductor, Inc., CA, USA.

His research interests include model compression and efficient AI systems.



Weidong Shao received the M.S. degree from Stanford University, CA, USA, in 2009. He was Senior AI Tools and Data Platform Director at Black Sesame Technologies, Inc., CA, USA, from 2020 to 2024. He is currently an independent consultant specializing in system integration and AI/ML solutions, helping organizations transition cutting-edge research with production-grade deployments.

Anuj Pathania received his Ph.D. degree from Karlsruhe Institute of Technology (KIT), Germany in 2018.

He is currently an Assistant Professor at the University of Amsterdam. His research focuses on the high-performance, low-power design of embedded systems.

Tulika Mitra received a BE degree in computer science from Jadavpur University, Kolkata, India, in 1995, an ME degree in computer science from the Indian Institute of Science, Bengaluru, India, in 1997, and a Ph.D. degree from the State University of New York, Stony Brook, NY, USA, in 2000. She is currently Provost's Chair Professor of Computer Science at the School of Computing, National University of Singapore, Singapore. Her research interests include the design automation of embedded realtime systems with particular emphasis on software timing analysis/optimizations, application-specific processors, energy-efficient computing, and heterogeneous computing.