MOSAIC: Heterogeneity-, Communication-, and Constraint-Aware Model Slicing and Execution for Accurate and Efficient Inference

Myeonggyun Han
UNIST
hmg0228@unist.ac.kr

Jihoon Hyun
UNIST
jhyun0812@unist.ac.kr

Seongbeom Park
UNIST
amita90@unist.ac.kr

Jinsu Park
UNIST
jinsupark@unist.ac.kr

Woongki Baek
UNIST
wbaek@unist.ac.kr

***Abstract*—Heterogeneous embedded systems have surfaced as a promising solution for accurate and efficient deep-learning inference on mobile devices. Despite extensive prior works, it still remains unexplored to investigate the system-software support that efficiently executes inference workloads by judiciously considering their performance and energy heterogeneity, communication overheads, and constraints.**

To bridge this gap, we propose MOSAIC, heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference on heterogeneous embedded systems. MOSAIC generates the efficient model slicing and execution plan for the target inference workload through dynamic programming. MOSAIC significantly reduces inference latency and energy, exhibits high estimation accuracy, and incurs small overheads.

***Keywords*-Model Slicing and Execution; Inference; Heterogeneous Embedded Systems;**

I. Introduction

The need for accurate and efficient deep-learning inference on mobile systems is ever increasing to enable intelligent and interactive services such as augmented reality and personal mobility. For a wide range of mobile applications such as security- and privacy-sensitive applications, it is highly crucial to execute inference workloads within mobile systems without relying on cloud services, which may leak sensitive information through various security attacks.

Heterogeneous embedded systems are rapidly emerging as a promising solution to enable accurate and efficient inference on mobile systems [1, 2, 3, 4, 5]. Heterogeneous embedded systems comprise various computing devices (e.g., big core cluster, little core cluster, GPU, and neural processing unit (NPU)), which exhibit widely-different characteristics in terms of performance, energy consumption, functionality (e.g., supported operations), memory capacity, and communication overheads.

Inference workloads also exhibit widely-different characteristics in terms of heterogeneity in performance, energy efficiency, communication overheads, and constraints across computing devices on heterogeneous embedded systems. Despite extensive prior works, it still remains unexplored to investigate the system-software support that efficiently executes inference workloads on heterogeneous embedded systems by judiciously considering their characteristics.

To bridge this gap, this work proposes MOSAIC, heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference on heterogeneous embedded systems. MOSAIC builds on the accurate models for estimating the execution and communication costs, generates the efficient model slicing and execution plan with low time complexity, and executes the target inference workload to significantly improve its efficiency based on the user-specified metric such as latency and energy.

Specifically, this paper makes the following contributions:

- We propose MOSAIC, a software-based system for heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference on heterogeneous embedded systems. MOSAIC employs the accurate models for estimating the execution and communication costs of the target inference workload. MOSAIC generates the efficient model slicing and execution plan for the target workload using an algorithm based on dynamic programming.
- We design and implement the prototype of MOSAIC as a user-level runtime system using the TensorFlow Lite programming framework [47] for deep-learning inference on the Android OS. MOSAIC achieves high efficiency by executing the slices of the target inference workload across computing devices on the underlying heterogeneous embedded system in a heterogeneity-, communication-, and constraint-aware manner.
- We quantify the effectiveness of MOSAIC with widely-used inference workloads on a full heterogeneous embedded system that comprises state-of-the-art big core cluster, little core cluster, GPU, and NPU. Our experimental results demonstrate the effectiveness of MOSAIC as it significantly improves the efficiency of inference (e.g., 29.2% lower inference latency than an NPU-preferred version (i.e., `TF-NPU-P`) with the performance governor and large models and 36.6% lower energy consumption than an NPU-preferred version (i.e., `TF-NPU-O`) with the on-demand governor and large models), achieves high estimation accuracy, and incurs small overheads.

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00021

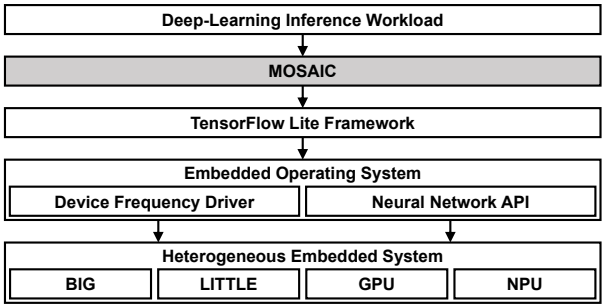


Figure 1: Hardware and software stacks for deep-learning inference on heterogeneous embedded systems.

The rest of this paper is organized as follows. Section II provides the background information for this work. Section III describes the experimental methodology. Section IV motivates the need for heterogeneity-, communication-, and constraint-aware inference. Section V discusses the design and implementation of MOSAIC. Section VI quantifies the effectiveness of MOSAIC. Section VII summarizes related work. Section VIII concludes the paper.

II. Background: Inference on Heterogeneous Embedded Systems

Heterogeneous embedded systems comprise various computing devices (e.g., big core cluster, little core cluster, GPU, NPU), which exhibit widely-different characteristics in terms of functionality, performance, energy efficiency, communication overheads, and memory capacity [1, 2, 3, 4, 5]. Representative programming frameworks for deep-learning inference on mobile systems such as TensorFlow Lite [47] provide simple abstraction for heterogeneous embedded systems in that computations can be offloaded to one of the devices by simply invoking their API functions without the need for writing device-specific code.

Figure 1 shows the hardware and software stacks for deep-learning inference on heterogeneous embedded systems. The grey component in Figure 1 is the system (i.e., MOSAIC) proposed in this work.

Deep-learning inference workloads consist of *layers*. Each layer comprises a set of associated mathematical operations (e.g., convolution, rectifier, and softmax). Each layer takes a set of input tensors (i.e., multidimensional arrays), performs computations that are specified by its operations, and generates a set of output tensors.

We refer a *model slice* as a set of consecutive layers that are executed on the same computing device on the underlying heterogeneous embedded system. There exist communication overheads between consecutive slices as they need to communicate through input and output tensors.

Model slices impose different memory and/or functionality constraints. Some of the computing devices on the underlying heterogeneous embedded system may be incapable of executing certain slices due to such constraints. For instance, if the size of a model slice exceeds the memory capacity of a computing device or a model slice includes mathematical operations that are unsupported by a computing device, the slice cannot be executed on the device.

III. Experimental Methodology

To investigate the characteristics of deep-learning inference workloads and the effectiveness of MOSAIC, we use a heterogeneous embedded system, the HiKey 970 embedded development board [6]. The evaluated system is equipped with the Kirin 970 mobile processor [1] that comprises a CPU including four Cortex-A73 (big) cores, four Cortex-A53 (little) cores, a Mali-G72 GPU, and a NPU. The big core cluster, little core cluster, and GPU support DVFS. The available frequency ranges of the big core cluster, little core cluster, and GPU are 682–2362MHz, 509–1844MHz, and 104–767MHz, respectively. The NPU lacks DVFS support.

The NPU has a memory constraint in that it cannot execute a model slice whose size exceeds 100MB [6]. While the exact memory constraints of the big core cluster, little core cluster and GPU for executing inference workloads on the evaluated system are undocumented, they are sufficiently large for the evaluated inference workloads.

As for the system software stack, the evaluated heterogeneous embedded system is installed with Android 8.1. In addition, all the evaluated inference workloads and MOSAIC are implemented using the TensorFlow Lite 1.11.0 [47].

Table I shows the inference workloads (i.e., Inception V4 (IN) [45], MnasNet with the model width parameters (p_W) of 1.0 (MN-1.0) and 1.3 (MN-1.3) [46], MobileNet V2 with $p_W = 1.3$ (MO-1.3) and $p_W = 1.4$ (MO-1.4) [40], ResNet V2 (RN) [19], VGG (VGG) [42]) with large models. They exhibit high accuracy (i.e., the Top-1 accuracy with the ImageNet [39] dataset) and widely-different characteristics such as the model size (i.e., the memory used by the model, which is reported by TensorFlow Lite), layer count, and inference latency (on the evaluated GPU). The evaluated workload set includes the state-of-the-art inference workloads (e.g., MobileNet V2 [40], MnasNet [46]), which are highly optimized for mobile systems. MobileNet V2 and MnasNet provide a mechanism to exploit the tradeoff between inference accuracy and latency through hyperparameters. Specifically, the model width parameter (i.e., p_W) determines the number of channels, which generally results in higher accuracy and longer latency when it is set to a larger value.

As shown in Table I, we use MnasNet with $p_W = 0.5$ (MN-0.5), MobileNet V2 with $p_W = 1.0$ (MO-1.0), and SqueezeNet (SN) [23] to investigate the impact of MOSAIC with smaller models. Due to the use of smaller models, they tend to exhibit lower accuracy.

To measure the latency of inference workloads, we use the `high_resolution_clock` function in the C++ standard library. To measure the energy consumption of inference workloads, we use an external power monitor [20], which

Table I: Evaluated deep-learning inference workloads

Workload	Accuracy	Input Size[1]	Model Size	Layers	Latency
Inception V4 (IN) [45]	80.1%	(1, 299, 299, 3)	183.7MB	20	430.0ms
MnasNet with $p_W = 1.0$ (MN-1.0) [46]	74.1%	(1, 224, 224, 3)	321.9MB	20	66.8ms
MnasNet with $p_W = 1.3$ (MN-1.3) [46]	75.2%	(1, 224, 224, 3)	511.6MB	20	81.3ms
MobileNet V2 with $p_W = 1.3$ (MO-1.3) [40]	74.4%	(1, 224, 224, 3)	187.8MB	18	49.1ms
MobileNet V2 with $p_W = 1.4$ (MO-1.4) [40]	75.0%	(1, 224, 224, 3)	214.7MB	18	55.4ms
ResNet V2 (RN) [19]	77.8%	(1, 224, 224, 3)	260.4MB	53	603.8ms
VGG (VGG) [42]	71.5%	(1, 224, 224, 3)	407.4MB	16	218.7ms
MnasNet with $p_W = 0.5$ (MN-0.5) [46]	68.0%	(1, 224, 224, 3)	98.3MB	20	42.7ms
MobileNet V2 with $p_W = 1.0$ (MO-1.0) [40]	71.8%	(1, 224, 224, 3)	94.5MB	18	36.6ms
SqueezeNet (SN) [23]	49.0%	(1, 224, 224, 3)	34.8MB	10	32.5ms

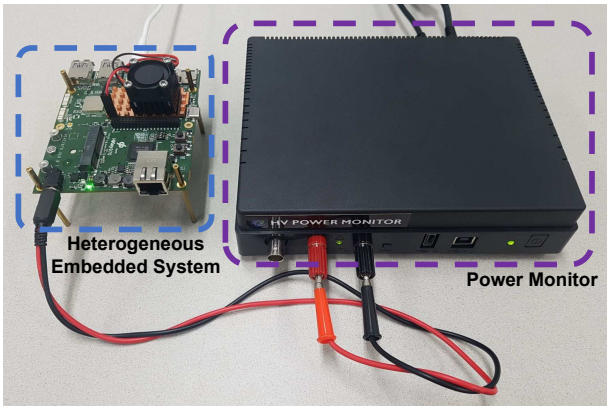


Figure 2: Evaluated heterogeneous embedded system and power monitor

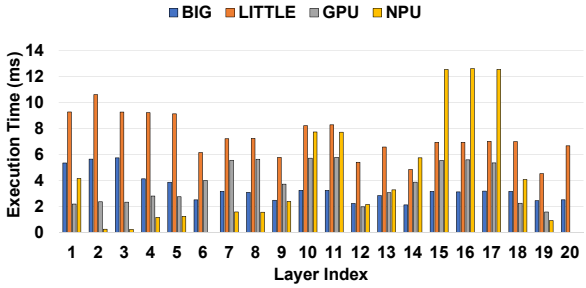
collects the voltage and current applied to the evaluated heterogeneous embedded system at the data sampling rate of 5000 samples per second. Figure 2 shows the power monitor connected to the evaluated heterogeneous embedded system.

IV. Need for Heterogeneity-, Communication-, and Constraint-Aware Inference

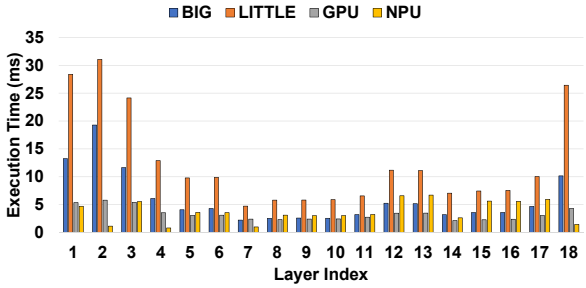
We investigate the characteristics of widely-used deep-learning inference workloads in terms of the model size, performance and energy heterogeneity, and communication overheads on the evaluated heterogeneous embedded system. For conciseness, we mainly report the data with MN-1.0 and MO-1.4, which represent accurate and highly-optimized inference workloads on mobile systems.

As shown in Table I, accurate inference workloads including the ones (e.g., MN-1.3, MO-1.4) highly optimized for mobile systems employ large models, which exceed the memory constraint of the NPU (i.e., 100MB) on the evaluated heterogeneous embedded systems. Further, the evaluated inference workloads exhibit widely-different model sizes, which indicates that different numbers of slices are required to execute them using the NPU.

[1]The first, second, third, and fourth elements in the tuples denote the batch size, height, width, and number of channels, respectively.



(a) MnasNet 1.0

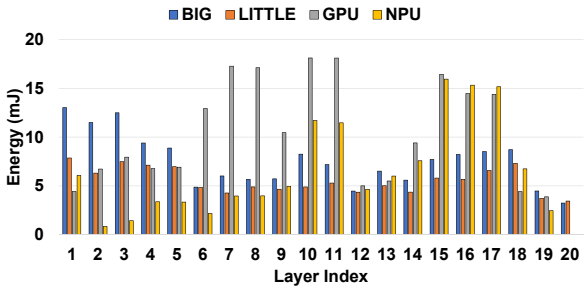


(b) MobileNet V2 1.4

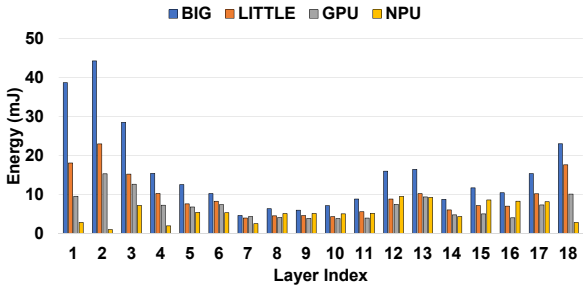
Figure 3: Performance heterogeneity of inference workloads

Figure 3 shows the execution time of each layer of MN-1.0 and MO-1.4 when it is executed on the big core cluster, little core cluster, GPU, and NPU. We observe that layers of MN-1.0 and MO-1.4 exhibit widely different performance characteristics across the devices. For instance, the GPU achieves significantly higher performance than the NPU when executing the layers 12–17 of MO-1.4. In contrast, the NPU significantly outperforms the GPU when executing the layers 2–9 of MN-1.0.

Figure 4 shows the energy consumption of each layer of MN-1.0 and MO-1.4 when it is executed on the big core cluster, little core cluster, GPU, and NPU. Similarly to performance heterogeneity, we also observe that layers of MN-1.0 and MO-1.4 exhibit widely different energy consumption characteristics across the devices. For example, the little core cluster consumes significantly lower energy than the NPU when executing the layers 9–17 of MN-1.0. The little core cluster tends to achieve higher efficiency



(a) MnasNet 1.0



(b) MobileNet V2 1.4

Figure 4: Energy heterogeneity of inference workloads

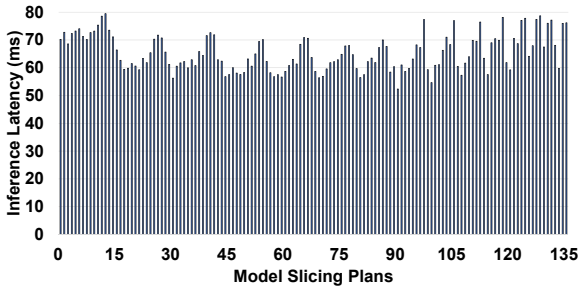


Figure 5: Communication overheads

when executing layers with lower computational intensity. In contrast, the NPU significantly outperforms the little core cluster in terms of energy efficiency when executing the layers 1–7 of MO-1.4.

Figure 5 shows the inference latency of MO-1.4 when they are decomposed into three slices with various slicing plans and the preferred computing device of each slice is set to the NPU. We observe that their inference latency is highly sensitive to the slicing plan. For instance, the performance difference of the best and worst slicing plans is 34.1%, which is significant. This data trend indicates that communication overheads have a significant impact on the efficiency of the target inference workload.

Overall, our experimental results show that the evaluated inference workloads exhibit widely-different characteristics in terms of the model size, performance and energy heterogeneity, and communication overheads. Therefore, it is crucial to investigate the system-software support for heterogeneity-, communication-, and constraint-aware model slicing and execution to achieve the best possible efficiency

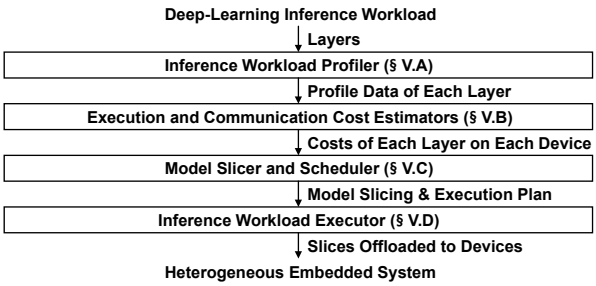


Figure 6: Overall architecture of MOSAIC

of inference workloads on heterogeneous embedded systems.

V. Design and Implementation

MOSAIC is a software-based system that determines the efficient model slicing and execution plan for the target deep-learning inference workload based on the user-defined metrics such as latency and energy consumption. Figure 6 shows the overall architecture of MOSAIC, which mainly consists of the inference workload profiler, the execution and communication cost estimators, the model slicer and scheduler, and the inference workload executor.

A. Inference Workload Profiler

The inference workload profiler of MOSAIC executes each of the layers in the target inference workload and profiles the total costs (e.g., latency, energy consumption) for executing each layer on computing devices in the underlying heterogeneous embedded system. If a computing device supports DVFS, the total costs for executing the layer are collected at two frequencies (i.e., the maximum and minimum frequencies available on the computing device).

If the layer cannot be executed on a certain computing device due to a constraint (e.g., memory constraint), the total cost for executing the layer on the device is set to an infinite value. It also collects the sizes of the input and output tensors of the layer, which are used to estimate the communication costs of the layer.

B. Execution and Communication Cost Estimators

The total cost of each layer measured by the inference workload profiler includes both the execution and communication costs associated with the layer. Based on the total cost and the input and output tensor sizes of each layer, the execution and communication cost estimators of MOSAIC estimate the execution and communication costs of the layer on each computing device in the heterogeneous embedded system.

1) Communication Cost Estimator

To investigate the relationship between the communication cost and the tensor size, we developed a microbenchmark that consists of layers with various tensor sizes. Figure 7 shows the communication time with various tensor sizes on the GPU (at its maximum frequency) and NPU. The communication time data with the CPU are omitted as our

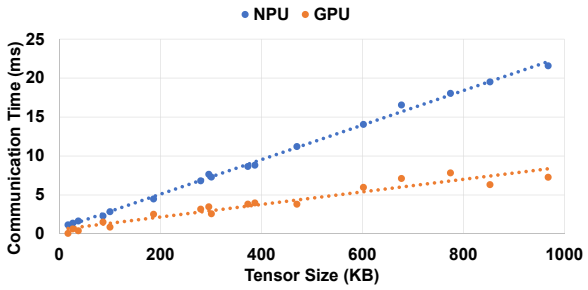


Figure 7: Communication time with various tensor sizes

experimental results show that they are insignificant due to no or small data copy and format transformation overheads. We observe the following data trends.

First, on both the GPU and NPU, the communication time is linearly proportional to the size of the tensors associated with the layer. Second, the NPU incurs significantly larger communication overheads than the GPU. While the implementation details of the evaluated NPU are undisclosed [1], we conjecture that the data copy and/or format transformation overheads for the NPU are significantly larger than those for the GPU.

Guided by the aforementioned observations, we design and implement the communication cost estimator based on the linear regression technique. Specifically, the communication cost estimator employs Equations 1 and 2 to estimate the communication costs associated with the input and output tensors of the layer l, where $T_{in,l}$, $T_{out,l}$, α_{d,f_d}, and β_{d,f_d} denote the total sizes of the input and output tensors of the layer l and the regression coefficients for a computing device d (i.e., the GPU or NPU) at frequency f_d.

$$c_{in,l,d,f_d} = \alpha_{d,f_d} \cdot T_{in,l} + \beta_{d,f_d} \quad (1)$$

$$c_{out,l,d,f_d} = \alpha_{d,f_d} \cdot T_{out,l} + \beta_{d,f_d} \quad (2)$$

2) Execution Cost Estimator

The execution cost estimator estimates the cost for executing each layer without including the communication costs. Specifically, the execution cost estimator first estimates the total execution cost of each layer on a computing device d running at f_d based on the profile data collected at the maximum and minimum frequencies of d. The execution cost estimator then simply subtracts the communication costs of the layer estimated by the communication cost estimator from the estimated total cost of the layer. The execution cost estimator consists of the performance and power estimators.

Performance Estimator: The performance estimator of the execution cost estimator estimates the latency for executing a layer l on a computing device d, which runs at frequency f_d. The performance estimator builds on a linear model, which is shown in Equation 3, where t_{l,d,f_d}, $\gamma_{l,d}$, and $\epsilon_{l,d}$ denote the estimated latency and coefficients. The coefficients (i.e., $\gamma_{l,d}$, $\epsilon_{l,d}$) are computed based on the profile data collected by the inference workload profiler at the maximum and minimum frequencies of d.

Table II: Voltage and frequency levels of the evaluated computing devices

Device	Voltage and frequency levels
Big cores	(0.7V, 682MHz), (0.8V, 1018MHz), (0.8V, 1210MHz), (0.8V, 1364MHz), (0.9V, 1498MHz), (0.9V, 1652MHz), (0.9V, 1863MHz), (1.0V, 2093MHz), (1.1V, 2362MHz)
Little cores	(0.7V, 509MHz), (0.8V, 1018MHz), (0.9V, 1210MHz), (0.9V, 1402MHz), (1.0V, 1556MHz), (1.0V, 1690MHz), (1.1V, 1844MHz)
GPU	(0.6V, 104MHz), (0.7V, 151MHz), (0.7V, 237MHz), (0.7V, 332MHz), (0.8V, 415MHz), (0.8V, 550MHz), (0.9V, 667MHz), (1.0V, 767MHz)

$$t_{l,d,f_d} = \frac{\gamma_{l,d}}{f_d} + \epsilon_{l,d} \quad (3)$$

Power Estimator: The power estimator of the execution cost estimator estimates the power consumption for executing a layer l on a computing device d running at frequency f_d. As shown in Equation 4, the power estimator decomposes the total power consumption into the dynamic (i.e., $P_{\text{dynamic},l,d,f_d}$) and static (i.e., P_{static,d,f_d}) power consumption. Since static power consumption is the inherent property of a computing device and independent of the characteristics of the target inference workload, we only profile it once for each computing device (i.e., no need for per-workload profiling).

$$P_{l,d,f_d} = P_{\text{dynamic},l,d,f_d} + P_{\text{static},d,f_d} \quad (4)$$

Since dynamic power consumption is dependent on not only the characteristics of the computing device (and its frequency) but also the characteristics of the layer, the power estimator estimates dynamic power consumption to eliminate the need for extensive offline profiling. Specifically, the power estimator employs Equation 5 to estimate the dynamic power consumption (i.e., $P_{\text{dynamic},l,d,f_d}$) of layer l running on computing device d at frequency f_d (and the corresponding voltage level V_{f_d}), where $f_{d,\text{max}}$, $V_{f_{d,\text{max}}}$, and $P_{\text{dynamic},l,d,f_{d,\text{max}}}$ denote the maximum frequency of d, the corresponding voltage level, and the dynamic power consumption at the maximum frequency collected by the inference workload profiler.

$$P_{\text{dynamic},l,d,f_d} = \frac{V_{f_d}^2 \cdot f_d}{V_{f_{d,\text{max}}}^2 \cdot f_{d,\text{max}}} \cdot P_{\text{dynamic},l,d,f_{d,\text{max}}} \quad (5)$$

The voltage and frequency levels of computing devices are readily available through their specifications or direct measurements. Table II shows the voltage and frequency levels of each computing device on the evaluated heterogeneous embedded system, which are publicly available their device tree source (DTS) files. Dynamic power consumption can

be estimated by plugging in specific values of f_d and V_{fd} in Equation 5.

C. Model Slicer and Scheduler

The main goal of the model slicer and scheduler (MSS) of MOSAIC is to generate the efficient model slicing and execution plan for the target deep-learning inference workload on the heterogeneous embedded system. Specifically, MSS determines the number of slices, the layers that belong to each slice, and the computing device that executes each slice in order to maximize the efficiency of the target inference workload on the heterogeneous embedded system based on the user-defined metric (e.g., latency, energy consumption).

We formulate the model slicing and execution problem as a dynamic-programming problem [11]. Without loss of generality, we assume that the target inference workload employs a deep-learning model that consists of Λ layers. $C_{m,n,\delta}$ denotes the total cost for executing the $n-m+1$ consecutive layers from the m-th layer to the n-th layer of the model on the computing device δ, where $1 \leq m \leq n \leq \Lambda$, $\delta \in \mathbb{D}$. Note that we consider same physical computing devices running at different frequencies as different computing devices to simplify the problem formulation of energy optimization.

In case of performance optimization, $\mathbb{D}$ is defined as Equation 6, where $B_{f_{B,\max}}$, $L_{f_{L,\max}}$, $G_{f_{G,\max}}$, N denote the big core cluster at its maximum frequency, the little core cluster at its maximum frequency, the GPU at its maximum frequency, and the NPU, respectively. Note that the NPU on the evaluated heterogeneous embedded system lacks the support for DVFS. On the evaluated system, $|\mathbb{D}|$ is 4 for performance optimization.

$$\mathbb{D} = \{B_{f_{B,\max}}, L_{f_{L,\max}}, G_{f_{G,\max}}, N\} \tag{6}$$

In case of energy optimization, $\mathbb{D}$ is defined as Equation 7, where $B_{f_{B,\min}}, B_{f_{B,\min+1}}, \cdots, B_{f_{B,\max}}$ denote the big core cluster at its minimum, second minimum, $\cdots$, and maximum frequencies, $L_{f_{L,\min}}, L_{f_{L,\min+1}}, \cdots, L_{f_{L,\max}}$ indicate the little core cluster at its minimum, second minimum, $\cdots$, and maximum frequencies, $G_{f_{G,\min}}, G_{f_{G,\min+1}}, \cdots, G_{f_{G,\max}}$ denote the GPU at its minimum, second minimum, $\cdots$, and maximum frequencies, and N indicates the NPU. On the evaluated system, $|\mathbb{D}|$ is 25 for energy optimization (see Table II).

$$\begin{aligned}\mathbb{D} = \{&B_{f_{B,\min}}, B_{f_{B,\min+1}}, \cdots, B_{f_{B,\max}},\\ &L_{f_{L,\min}}, L_{f_{L,\min+1}}, \cdots, L_{f_{L,\max}},\\ &G_{f_{G,\min}}, G_{f_{G,\min+1}}, \cdots, G_{f_{G,\max}},\\ &N\}\end{aligned} \tag{7}$$

$C_{m,n,\delta}$ is computed using Equation 8, where $e_{k,\delta}$, $c_{in,m,\delta}$, and $c_{out,n,\delta}$ denote the execution cost of the k-th layer ($m \leq k \leq n$) and the communication cost associated with the input tensors of the m-th layer, and the communication cost associated with the output tensors of the n-th layer, respectively. MSS employs the execution and communication cost estimators to estimate $e_{k,\delta}$, $c_{in,m,\delta}$, and $c_{out,n,\delta}$. If there is any layer that cannot be executed on δ due to a constraint, $C_{m,n,\delta}$ is set to an infinite value to avoid selecting the corresponding model slicing and execution plan.

$$C_{m,n,\delta} = \begin{cases} \sum_{k=m}^{n} e_{k,\delta} + c_{in,m,\delta} + c_{out,n,\delta} & \text{if } \delta \text{ can execute} \\ \infty & \text{otherwise} \end{cases} \tag{8}$$

$C_{tot,l}$ denotes the total cost to execute the consecutive layers from the first layer to the l-th layer. To formulate the model slicing and execution problem as a dynamic-programming problem, $C_{tot,l}$ must exhibit the optimal substructure and overlapping subproblem properties [11] in that $C_{tot,l}$ can be efficiently computed if $C_{tot,0}$, $C_{tot,1}$, $\cdots$, $C_{tot,l-1}$ are known. We consider all the possible cases, in each of which the l-th layer belongs to a unique slice. Since each layer belongs to only one slice and each slice comprises consecutive layers, there are l cases in total, where the l-th layer belongs to unique slices. Specifically, the slice that contains the l-th layer may comprise only a single layer (i.e., the l-th layer), two layers (i.e., the l-th layer and the $(l-1)$-th layer), $\cdots$, or l layers (i.e., the l-th layer, the $(l-1)$-th layer, $\cdots$, and the first layer).

Without loss of generality, we assume that the slice that contains the l-th layer comprises $l-k$ layers (i.e., the l-th, $(l-1)$-th, $\cdots$, and $(k+1)$-th layers). In this case, the lowest cost for executing the consecutive layers from the first layer to the l-th layer is computed by summing $C_{tot,k}$ and the total cost for executing the last $l-k$ layers on the device that incurs the minimum total cost among all the computing devices on the heterogeneous embedded system.

As shown in Equation 9, $C_{tot,l}$ can be then computed by finding the minimum total cost among all the l aforementioned cases. Since the model slicing and execution problem has the optimal substructure and overlapping subproblem properties, its optimal solution can be determined based on dynamic programming.

$$C_{tot,l} = \begin{cases} 0 & \text{if } l = 0 \\ \min\limits_{0 \leq k < l, \forall \delta \in \mathbb{D}} (C_{tot,k} + C_{k+1,l,\delta}) & \text{otherwise} \end{cases} \tag{9}$$

Algorithm 1 shows the pseudocode for the `findEfficientSlicingAndExecutionPlan` function that determines the efficient model slicing and execution plan based on dynamic programming. In the outer loop (Lines 4–19), MSS iterates the layer count from 1 to Λ. MSS uses the solutions found in previous iterations to find the solution for the current iteration in the outer loop.

In the inner loop (Lines 8–17), MSS iterates all the cases, in each of which the last layer belongs to a unique slice. MSS determines the set of slices that minimizes the total cost for executing the consecutive layers and memoizes its cost and slicing and execution plan to reuse them in the next iteration in the outer loop.

The proposed algorithm has low time complexity (i.e., $O(\Lambda^2 \cdot |\mathbb{D}|)$, where Λ and $|\mathbb{D}|$ denote the number of layers in the inference workload and the number of computing devices on the heterogeneous embedded system, respectively). As

Algorithm 1 The findEfficientSlicingAndExecutionPlan function

```
1: procedure FINDEFFICIENTSLICINGANDEXECUTIONPLAN(layers, devices)
2:     sliceSets[0] ← ∅
3:     sliceSets[0].cost ← 0
4:     for l ← 1 to layers.length do                                  ▷ layers.length = Λ
5:         sliceSet ← ∅
6:         sliceSet.cost ← ∞
7:         slice ← createNewSlice()
8:         for k ← 0 to (l − 1) do
9:             slice.layers ← getConsecutiveLayers(layers, k + 1, l)
10:            for δ in devices do                                    ▷ devices = 𝔻
11:                if δ.canExecute(slice) = true then
12:                    slice.device ← δ
13:                    cost ← sliceSets[k].cost + estimateTotalCost(slice)
14:                    if cost < sliceSet.cost then
15:                        sliceSet ← sliceSets[k]
16:                        sliceSet.insert(slice)
17:                        sliceSet.cost ← cost
18:        sliceSets[l] ← sliceSet
19:        sliceSets[l].cost ← sliceSet.cost
20:    return sliceSets[layers.length]
```

quantified in Section VI, MOSAIC achieves high inference efficiency with small overheads due to the use of an efficient algorithm.

D. Inference Workload Executor

The inference workload executor of MOSAIC is a user-level runtime system that executes the model slices of the target inference workload across the computing devices on the heterogeneous embedded system based on the efficient model slicing and execution plan generated by MSS. The current version of the inference workload executor is implemented in the C++ programming language based on the TensorFlow Lite framework [47] on the Android OS. However, we believe that MOSAIC is readily applicable to other widely-used deep-learning frameworks as it builds on a framework-agnostic approach for slicing and executing the target inference workload.

VI. Evaluation

A. Overview

This section quantifies the effectiveness of MOSAIC. Specifically, we aim to investigate (1) inference latency, (2) inference energy, (3) impact of the MOSAIC components, (4) efficiency with smaller models, (5) estimation accuracy, and (6) overheads for generating the model slicing and execution plan.

For each inference workload, we evaluate ten versions – the big core cluster-preferred (`TF-BIG-P`), little core cluster-preferred (`TF-LITTLE-P`), GPU-preferred (`TF-GPU-P`), NPU-preferred (`TF-NPU-P`) with the performance governor of the Android OS, the big core cluster-preferred (`TF-BIG-O`), little core cluster-preferred (`TF-LITTLE-O`), GPU-preferred (`TF-GPU-O`), NPU-preferred (`TF-NPU-O`) with the on-demand governor of the Android OS, exhaustive, and MOSAIC versions.

The big core cluster-, little core cluster-, GPU-, and NPU-preferred versions execute the slices of each inference workload on the corresponding preferred computing device. For the big core cluster-, little core cluster-, GPU-, and NPU-preferred versions, we include as many consecutive layers as possible in each slice if they satisfy all the constraints such as memory and functionality constraints. If a layer cannot be included in the slice that contains its previous layer due to a memory constraint, the layer is included in the next slice that is executed on the preferred device. If a layer cannot be executed on the preferred device due to a memory or functionality constraint, the layer is executed in a separate slice on the NPU (if feasible), GPU (if feasible), or big core cluster (as the final fallback execution path).

As quantified by our experimental results, the performance governor tends to achieve higher performance than the on-demand governor as the performance governor always executes the target inference workload at the maximum frequency of the underlying computing device. In contrast, the on-demand governor, which is the default governor of the Android OS, tends to exhibit lower energy consumption than the performance governor as the on-demand governor performs DVFS based on the dynamic load of the target inference workload.

The exhaustive version uses the model slicing and execution plan with the highest inference efficiency (e.g., the lowest latency), which is empirically determined through exhaustive search. Note that it takes excessive computing time and resources to determine the model slicing and exe-

Table III: Model slicing and execution plans for performance optimization

Workload	Model slicing and execution plan
IN	N_1, N_{11}, B_{20}
MN-1.0	N_1, B_{10}, G_{18}, B_{20}
MN-1.3	N_1, B_9, G_{12}, B_{20}
MO-1.3	N_1, G_9, N_{18}
MO-1.4	N_1, G_8, N_{18}
RN	B_1, N_2, B_4, N_5, B_{12}, N_{13}, N_{31}, B_{48}, N_{49}, B_{52}
VGG	N_1, G_{14}, B_{16}

cution plan for the exhaustive version, which is impractical. For instance, it is estimated to take 1335 days to empirically determine the best model slicing and execution plan for MN-1.3 through exhaustive search using the evaluated system.

Due to limited computing time and resources, we determine the model slicing and execution plan for the exhaustive version of each inference workload by executing the workload with the 1000 unique model slicing and execution plans that are randomly selected and choosing the best plan among the randomly selected plans and the plans used by the other versions including the MOSAIC version. We report the results with the exhaustive version as the efficiency that can be potentially achieved by any competitive model slicing and execution technique.

Finally, the MOSAIC version uses MOSAIC to generate the efficient model slicing and execution plan.

B. Inference Latency

We investigate the effectiveness of MOSAIC in terms of inference latency. Figure 8 shows the inference latency of each version of the inference workloads with large models, normalized to the `TF-GPU-O` version that is the default setting of the Android OS. The rightmost bars show the average (i.e., geometric mean) inference latency of each version across the workloads. In addition, Table III shows the model slicing and execution plans generated by MOSAIC to minimize the latency of each workload. Each letter (i.e., big core cluster (B), little core cluster (L), GPU (G), and NPU (N)) indicates a slice and the device used to execute the slice. The subscript to each letter denotes the ID of the first layer of the slice.

First, MOSAIC significantly outperforms the big core cluster-, little core cluster-, GPU-, and NPU-preferred versions. Specifically, MOSAIC exhibits 70.3%, 86.1%, 39.1%, and 29.2% lower inference latency than the `TF-BIG-P`, `TF-LITTLE-P`, `TF-GPU-P`, and `TF-NPU-P` versions, respectively. MOSAIC significantly reduces inference latency by slicing and executing the model of the target inference workload in a heterogeneity-, communication-, and constraint-aware manner. For instance, as shown in Table III, MOSAIC effectively utilizes various computing devices (i.e., big core cluster, GPU, and NPU) when executing MN-1.3 and significantly reduces its inference latency.

Table IV: Model slicing and execution plans for energy optimization

Workload	Model slicing and execution plan
IN	N_1^{960}, N_{11}^{960}, L_{20}^{509}
MN-1.0	G_1^{767}, L_7^{509}, G_{12}^{667}, L_{15}^{509}, G_{18}^{667}, L_{20}^{509}
MN-1.3	G_1^{767}, L_7^{509}, G_{12}^{667}, L_{14}^{509}, G_{18}^{667}, L_{20}^{509}
MO-1.3	N_1^{960}, L_5^{509}, B_7^{682}, L_{10}^{1018}, L_{11}^{509}, L_{14}^{1018}, L_{15}^{509}, N_{18}^{960}
MO-1.4	N_1^{960}, L_5^{1018}, L_6^{509}, L_{10}^{1018}, L_{11}^{509}, L_{13}^{1402}, G_{14}^{767}, N_{18}^{960}
RN	B_1^{682}, N_2^{960}, B_4^{682}, N_5^{960}, L_{12}^{1018}, N_{13}^{960}, N_{29}^{960}, L_{48}^{1210}, N_{49}^{960}, L_{52}^{509}, B_{53}^{682}
VGG	N_1^{960}, G_{14}^{550}, G_{15}^{667}, B_{16}^{682}

MOSAIC exhibits the inference latency similar to that of the `TF-NPU-P` version with IN and RN. This is mainly because the NPU exhibits the highest performance among the computing devices and the model slicing plan used for the `TF-NPU-P` version happens to incur small communication overheads when executing IN and RN. Nevertheless, the NPU-preferred versions provide no guarantee for maximizing the efficiency across a wide range of inference workloads as they execute the target inference workload in a heterogeneity- and communication-oblivious manner.

Second, MOSAIC achieves the performance similar to that of the exhaustive version, which empirically determines the efficient model slicing and execution plan for the target inference workload through exhaustive search. Specifically, the average performance difference between the exhaustive and MOSAIC versions is 0.67% across the workloads, which is small. Note that the exhaustive version is guaranteed to exhibit (at least) the same efficiency as MOSAIC because we always include the model slicing and execution plan determined by MOSAIC in the plans that are explored by the exhaustive version. Our experimental results clearly demonstrate the effectiveness of MOSAIC in that it achieves high inference efficiency without the need for the exhaustive search process, which requires excessive computing time and resources.

C. Inference Energy

We investigate the effectiveness of MOSAIC in terms of inference energy. Figure 9 shows the energy consumption of each version of the inference workloads with large models, normalized to the `TF-GPU-O` version. The rightmost bars show the average (i.e., geometric mean) energy consumption across the workloads. In addition, Table IV shows the model slicing and execution plans generated by MOSAIC for energy optimization. The superscript to each letter denotes the frequency of the computing device in MHz.

First, MOSAIC significantly outperforms the big core cluster-, little core cluster-, GPU-, and NPU-preferred ver-

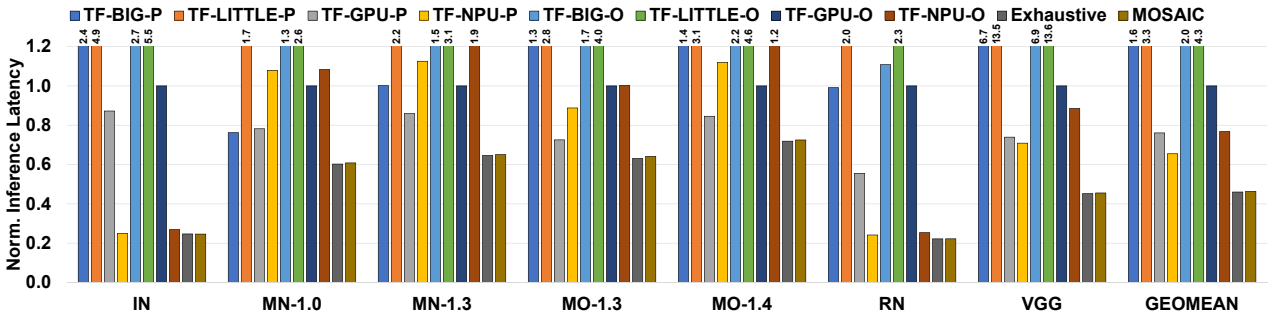


Figure 8: Inference latency

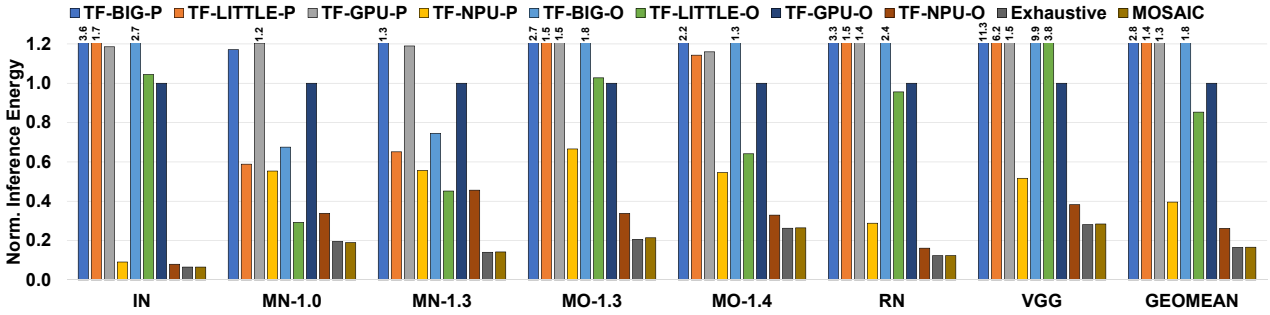


Figure 9: Inference energy

sions across the workloads in terms of energy efficiency. For instance, MOSAIC consumes 91.0%, 80.5%, 83.4%, and 36.6% lower energy than the `TF-BIG-O`, `TF-LITTLE-O`, `TF-GPU-O`, and `TF-NPU-O` versions, respectively. As shown in Table IV, MOSAIC effectively utilizes various computing devices at various frequencies, significantly reducing the energy consumption of the inference workloads.

Second, MOSAIC exhibits the energy consumption similar (i.e., the average difference of 0.53%) to the exhaustive version, which requires excessive computing time and resources. Our experimental results demonstrate that MOSAIC can be effectively used for both inference latency and energy optimizations on heterogeneous embedded systems.

D. Impact of the MOSAIC Components

We investigate the impact of the MOSAIC components in terms of inference latency and energy. To this end, we synthesize the heterogeneity- and constraint-aware (HCA) version, which is an intermediate version that generates the model slicing and execution plan by only considering the efficiency heterogeneity and constraints of each layer (i.e., in a communication-oblivious manner). We report the results with the HCA version to investigate the efficiency impact of heterogeneity- and constraint-aware model slicing and execution. Note that the efficiency difference between the HCA and MOSAIC versions shows the efficiency impact of communication-aware model slicing and execution.

Figure 10 shows the latency impact of the MOSAIC components. We observe that the HCA version considerably outperforms the `TF-GPU-O` version, which demonstrates the effectiveness of heterogeneity- and constraint-aware model slicing and execution. Further, MOSAIC considerably outperforms the HCA version, which demonstrates the impact of communication-aware model slicing and execution. The HCA version incurs higher inference latency than MOSAIC as it slices and executes the target inference workload in a communication-oblivious manner.

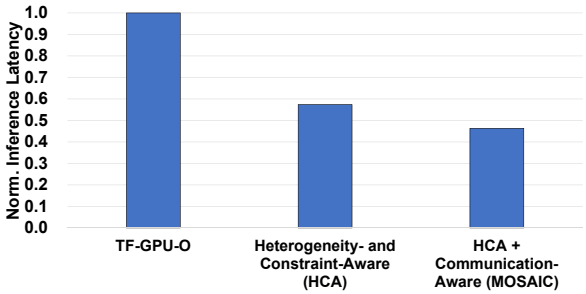


Figure 10: Latency impact of the MOSAIC components

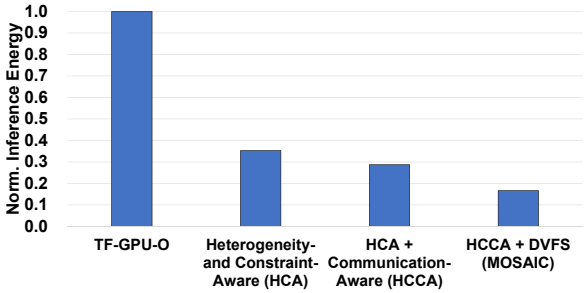


Figure 11: Energy impact of the MOSAIC components

We quantify the energy impact of the MOSAIC components. To this end, we synthesize an additional intermediate

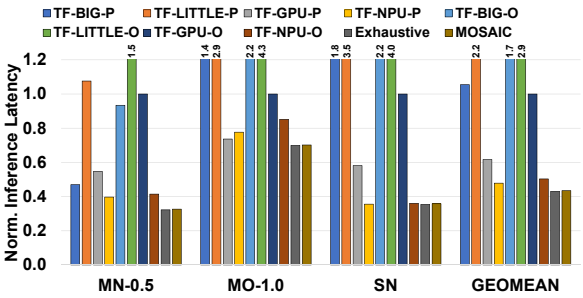


Figure 12: Inference latency with smaller models

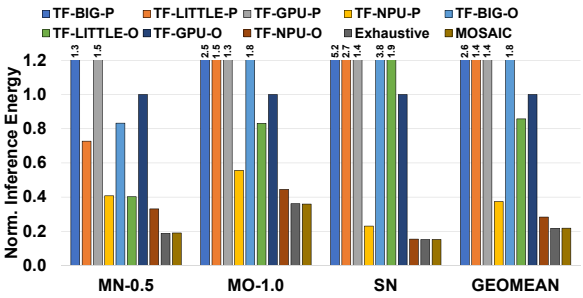


Figure 13: Inference energy with smaller models

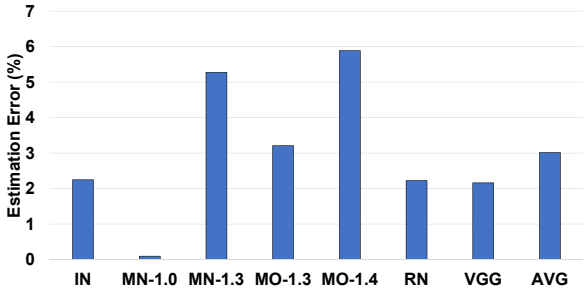


Figure 14: Latency estimation accuracy

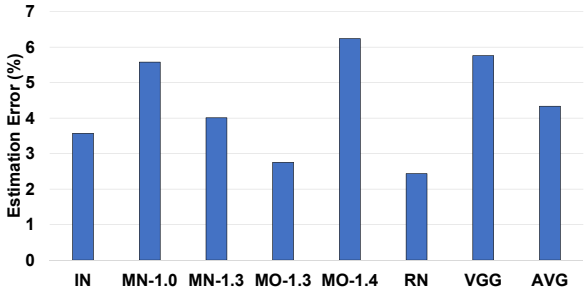


Figure 15: Energy estimation accuracy

version called the HCCA version, which generates the model slicing and execution plan by considering the heterogeneity of the physical computing devices, constraints of each layer, and communication overheads between devices in a DVFS-oblivious manner. Figure 11 shows the energy impact of the MOSAIC components.

The HCA version exhibits considerably lower energy consumption than the `TF-GPU-O` version, demonstrating the impact of heterogeneity- and constraint-aware model slicing and execution. In addition, the HCCA version consumes considerably lower energy than the HCA version, showing the effectiveness of communication-aware model slicing and execution. Finally, the MOSAIC version significantly exhibits significantly lower energy consumption than the HCCA version, which demonstrates the effectiveness of DVFS. In summary, our quantitative evaluation demonstrates that the individual components of MOSAIC compose in a constructive manner, providing additional performance and energy-efficiency gains.

E. Discussion

Smaller models: Figures 12 and 13 show the inference latency and energy consumption with smaller models. Our experimental results show that MOSAIC continues to achieve high efficiency with inference workloads with smaller models (i.e., MN-0.5, MO-1.0, SN). For instance, MOSAIC exhibits 58.8%, 80.4%, 29.5%, and 9.2% lower inference latency than the `TF-BIG-P`, `TF-LITTLE-P`, `TF-GPU-P`, and `TF-NPU-P` versions and performs similarly (i.e., the average difference of 0.89%) to the exhaustive version. Further, MOSAIC consumes 87.6%, 74.5%, 78.2%, and 22.9% lower energy than the `TF-BIG-O`, `TF-LITTLE-O`, `TF-GPU-O`, and `TF-NPU-O` versions and achieves similar energy efficiency (i.e., the average difference of 0.32%) to the exhaustive version. The efficiency gains of MOSAIC with smaller models decrease as fewer slices are generated, which reduces optimization opportunities.

Estimation Accuracy: Figures 14 and 15 show the latency and energy consumption estimation accuracy of MOSAIC. Our experimental results show that the latency and energy consumption estimation accuracy of MOSAIC is high. Specifically, the average latency and energy estimation errors are 3.0% and 4.3%, which are small.

Overheads: Figures 16 and 17 shows the overheads for performance and energy optimization. Our experimental results show that MOSAIC incurs small overheads for generating model slicing and execution plans. Specifically, the average times spent for generating the model slicing and execution plans across the workloads for performance and energy optimization are 0.15ms and 0.82ms, which are short. Since MOSAIC produces the model slicing and execution plan based on the efficient algorithm with low time complexity (i.e., $O(\Lambda^2 \cdot |\mathbb{D}|)$), it incurs insignificant overheads. Further, note that MOSAIC generates the model slicing and execution plan only once for each inference workload and incurs no overheads during the execution of the workload.

MOSAIC incurs larger overheads for energy optimization than performance optimization. This is mainly because the number of computing devices (i.e., $|\mathbb{D}|$) increases (from 4 (i.e., performance optimization) to 25 (i.e., energy optimization)) as MOSAIC considers each computing device at its maximum frequency for performance optimization but each

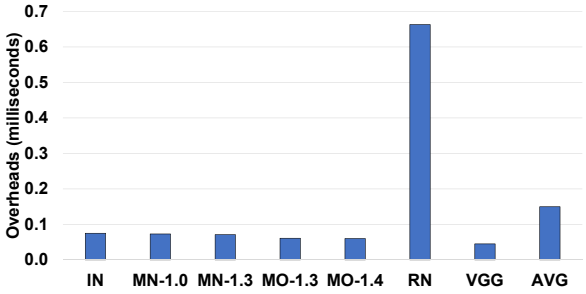


Figure 16: Overheads for performance optimization

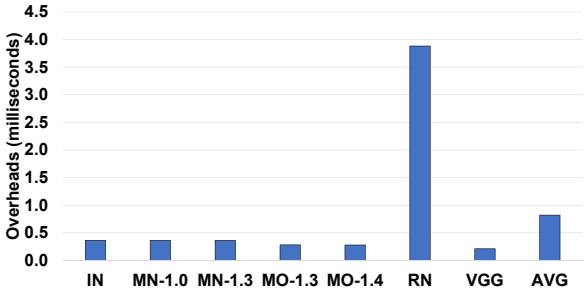


Figure 17: Overheads for energy optimization

computing device at all its available frequencies for energy optimization.

Overall, our quantitative evaluation shows the effectiveness of MOSAIC in that it significantly improves the efficiency of inference workloads in terms of latency and energy consumption, achieves high estimation accuracy, and incurs small overheads on the evaluated heterogeneous embedded system.

VII. Related Work

Prior works have investigated model analysis and/or optimization techniques to improve inference efficiency [7, 10, 21, 24, 26, 29, 37, 44] and address the memory constraints of deep-learning workloads [13, 38, 49]. While insightful, none of the prior works considers the efficiency heterogeneity, communication overheads, and constraints of inference workloads and emerging computing devices in an integrated manner.

The prior works proposed in [7, 10, 21, 24, 26, 29, 37, 44] lack the consideration of the efficiency heterogeneity and memory and functionality constraints of inference workloads and emerging computing devices (e.g., NPU), which are crucial factors to achieve the best possible efficiency on heterogeneous embedded systems. The works proposed in [13, 38, 49] lack the consideration of the efficiency heterogeneity, communication overheads, and functionality constraints of computing devices and deep-learning workloads. Our work significantly differs in the sense that it analyzes the characteristics of inference workloads on a state-of-the-art heterogeneous embedded system that includes a highly-optimized NPU, proposes an efficient algorithm to solve the model slicing and execution problem in a heterogeneity-, communication-, and constraint-aware manner, and designs, implements, and evaluates the proposed system using full hardware and software stacks.

Prior works have presented runtime techniques to efficiently execute training workloads for deep learning [15, 22, 30, 52]. The prior works lack heterogeneity-, communication-, and constraint-aware model slicing and execution in the presence of computing devices with functional and architectural heterogeneity. Our work differs in that it investigates the model slicing and execution technique to significantly improve the efficiency of inference workloads on heterogeneous embedded systems in a heterogeneity-, communication-, and constraint-aware manner.

Prior works have presented the design and implementation of hardware accelerators for deep learning [8, 9, 12, 14, 16, 18, 25, 32, 43, 50]. As quantified in this work based on a heterogeneous embedded system equipped with a deep-learning hardware accelerator (i.e., NPU), MOSAIC can be robustly used to effectively utilize the hardware accelerators along with general-purpose computing devices (e.g., CPU, GPU) by executing the slices of inference workloads in a heterogeneity-, communication-, and constraint-aware manner.

Prior works have investigated the architectural and system software techniques to improve the efficiency of heterogeneous computing systems [17, 27, 28, 31, 33, 34, 35, 36, 41, 48, 51]. Our work differs as it investigates the characteristics of various inference workloads on heterogeneous computing devices including a state-of-the-art NPU and presents a system that efficiently executes inference workloads on heterogeneous embedded systems.

VIII. Conclusions

This paper presents MOSAIC, heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference on heterogeneous embedded systems. MOSAIC uses the accurate models for estimating the execution and communication costs of the target inference workload and generates the efficient model slicing and execution plan with low time complexity. Our quantitative evaluation with the state-of-the-art inference workloads and heterogeneous embedded system shows that MOSAIC significantly reduces inference latency and energy (e.g., 29.2% lower inference latency than an NPU-preferred version (i.e., `TF-NPU-P`) with the performance governor and large models and 36.6% lower energy than an NPU-preferred version (i.e., `TF-NPU-O`) with the on-demand governor and large models), achieves high estimation accuracy, and incurs small overheads.

Acknowledgements

This research was partly supported by NRF (NRF-2016M3C4A7952587, NRF-2018R1C1B6005961) and IITP (No. 1711080972). Woongki Baek is the corresponding author.

REFERENCES

[1] http://www.hisilicon.com/en/Products/ProductList/Kirin.

[2] http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html.

[3] https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/.

[4] https://www.qualcomm.com/products/snapdragon-855-mobile-platform.

[5] https://www.apple.com/iphone-xs/a12-bionic/.

[6] https://www.96boards.org/product/hikey970/.

[7] A. Anderson and D. Gregg. "Optimal DNN Primitive Selection with Partitioned Boolean Quadratic Programming". In: *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 2018.

[8] S. Angizi, Z. He, and D. Fan. "DIMA: A Depthwise CNN In-Memory Accelerator". In: *Proceedings of the International Conference on Computer-Aided Design*. 2018.

[9] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic. "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019.

[10] S. Bateni, H. Zhou, Y. Zhu, and C. Liu. "PredJoule: A Timing-Predictable Energy Optimization Framework for Deep Neural Networks". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. 2018.

[11] R. E. Bellman. *Dynamic Programming*. 2003.

[12] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning". In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2014.

[13] X. Chen, D. Z. Chen, and X. S. Hu. "moDNN: Memory optimal DNN training on GPUs". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018.

[14] Y.-H. Chen, J. Emer, and V. Sze. "Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. 2016.

[15] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. 2014.

[16] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang. "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference". In: *Proceedings of the 55th Annual Design Automation Conference*. 2018.

[17] M. Han, J. Park, and W. Baek. "CHRT: A Criticality- and Heterogeneity-aware Runtime System for Task-parallel Applications". In: *Proceedings of the Conference on Design, Automation & Test in Europe*. 2017.

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. 2016.

[19] K. He, X. Zhang, S. Ren, and J. Sun. "Identity Mappings in Deep Residual Networks". In: *Computer Vision – ECCV 2016*. 2016.

[20] *High Voltage Power Monitor*. https://www.msoon.com/.

[21] L. N. Huynh, Y. Lee, and R. K. Balan. "DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications". In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 2017.

[22] J. Hyun, J. Park, K. Y. Kim, S. Yu, and W. Baek. "CEML: a Coordinated Runtime System for Efficient Machine Learning on Heterogeneous Computing Systems". In: *Euro-Par 2018: Parallel Processing*. 2018.

[23] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size". In: *CoRR* (2016).

[24] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon. "IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers". In: *Proceedings of the ACM Symposium on Cloud Computing*. 2018.

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 2017.

[26] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 2017.

[27] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction". In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. 2003.

[28] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. "Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era". In: *Proceedings of the 50th Annual Design Automation Conference*. 2013.

[29] Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, and J. W. Lee. "A Portable, Automatic Data Qantizer for Deep Neural Networks". In: *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. 2018.

[30] Y. Oyama, T. Ben-Nun, T. Hoefler, and S. Matsuoka. "Accelerating Deep Learning Frameworks with Micro-Batches". In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 2018.

[31] P. Pandit and R. Govindarajan. "Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices". In: *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*. 2014.

[32] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 2017.

[33] J. Park and W. Baek. "RCHC: A Holistic Runtime System for Concurrent Heterogeneous Computing". In: *2016 45th International Conference on Parallel Processing (ICPP)*. 2016.

[34] J. Park and W. Baek. "HAP: A Heterogeneity-Conscious Runtime System for Adaptive Pipeline Parallelism". In: *Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833*. 2016.

[35] 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.

[36] A. Prakash, S. Wang, A. Irimiea, and T. Mitra. "Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms". In: *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. 2015.

[37] H. Qi, E. R. Sparks, and A. Talwalkar. "Paleo: A Performance Model for Deep Neural Networks". In: *ICLR*. 2017.

[38] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler. "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-efficient Neural Network Design". In: *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. 2016.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vision* (2015).

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[41] A. Sethia and S. Mahlke. "Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution". In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 2014.

[42] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* (2014).

[43] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li. "C-brain: A Deep Learning Accelerator That Tames the Diversity of CNNs Through Adaptive Data-level Parallelization". In: *Proceedings of the 53rd Annual Design Automation Conference*. 2016.

[44] M. Song, Y. Hu, H. Chen, and T. Li. "Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017.

[45] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *AAAI*. 2017.

[46] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *ArXiv e-prints* (2018).

[47] *TensorFlow Lite*. https://www.tensorflow.org/lite/.

[48] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. "Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE)". In: *Proceedings of the 39th Annual International Symposium on Computer Architecture*. 2012.

[49] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska. "Superneurons: Dynamic GPU Memory Management for Training Deep Neural Networks". In: *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2018.

[50] X. Wang, J. Yu, C. Augustine, R. Iyer, and R. Das. "Bit Prudent In-Cache Acceleration of Deep Convolutional Neural Networks". In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2019.

[51] J. Yun, J. Park, and W. Baek. "HARS: A Heterogeneity-aware Runtime System for Self-adaptive Multithreaded Applications". In: *Proceedings of the 52Nd Annual Design Automation Conference*. 2015.

[52] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing. "Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters". In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017.