MapFormer: Attention-based multi-DNN Manager for Throughout & Power Co-Optimization on Embedded Devices

Andreas Karatzas
andreas.karatzas@siu.edu
School of Electrical, Computer and Biomedical Engineering
Southern Illinois University
Carbondale, U.S.A

Iraklis Anagnostopoulos
iraklis.anagno@siu.edu
School of Electrical, Computer and Biomedical Engineering
Southern Illinois University
Carbondale, U.S.A

ABSTRACT

In the context of modern services that use multiple Deep Neural Networks (DNNs), managing workloads on embedded devices presents unique challenges. These devices often incorporate diverse architectures, necessitating advanced management solutions to efficiently deploy multi-DNN workloads. Traditionally, the focus has been on improving throughput, while power optimization has received less attention. This paper presents MapFormer, a new manager that uses attention-based mechanisms to enhance both throughput and power efficiency. MapFormer intelligently assigns multi-DNN workloads to different computing components of embedded systems—CPU, GPU, and DLA—and adjusts operational frequencies to optimize power use. Experimental results show that MapFormer significantly improves average throughput under set power budgets by 90.8%, offering a promising approach for managing complex workloads on heterogeneous embedded systems.

KEYWORDS

Deep Neural Networks, Multi-DNN Workloads, Heterogeneous Architectures, Edge Inference, DNN Performance Prediction, Transformers, Throughput Optimization, Power Optimization

ACM Reference Format:
Andreas Karatzas and Iraklis Anagnostopoulos. 2024. MapFormer: Attention-based multi-DNN Manager for Throughout & Power Co-Optimization on Embedded Devices. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676724

1 INTRODUCTION

Deep Neural Networks (DNNs) have become core components for various mobile applications, such as malicious software detection, human activity monitoring, and medical health monitoring [34]. This has increased the demand for on-device deep learning (DL). On-device processing addresses privacy concerns and results in faster response time by skipping server-side steps, such as data uploading. However, on-device ML is not trivial [16]. More than half of new

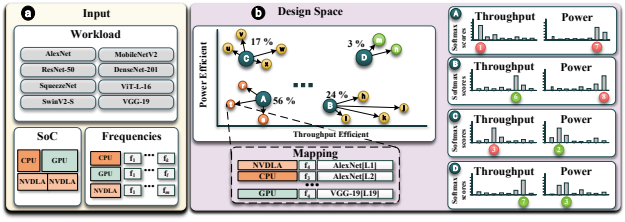

Figure 1: Motivational example. We explore a workload of 8 concurrently executed DNNs mapped randomly on the NVIDIA AGX Xavier board.

module devices include a deep learning accelerator (DLA) [27]. Nonetheless, modern deep learning frameworks often utilize the CPU, the GPU, or DLA, but not all of them synergistically [10]. This results in a dissatisfactory user experience.

Furthermore, runtime managers for embedded devices also encounter the challenge of serving multi-DNN workloads. For example, a mobile device may have to concurrently execute DL models for mobile visual tasks, user verification, mobile web browsing, and even help blind people discover their surroundings [19]. In such scenarios, the challenge of efficiently managing the available computing resources becomes crucial. One approach to address this issue is to assign DNNs to specific processing units, such as the CPU, embedded GPUs, or DLAs. However, these coarse-grained allocation strategies are suboptimal and leave considerable room for improvement [5]. To that end, a promising solution is splitting a DNN into multiple sub-DNNs and building pipelines, where each stage is offloaded to a different computing component [13].

Apart from throughput optimization, runtime managers must also explore the available power settings and select the frequencies that maximize power efficiency with minimum negative impact regarding throughput. This dimension adds significant complexity to the problem. To demonstrate this, we present an example of a multi-DNN workload that comprises 8 DNNs in Figure 1. In this example, we create 100 random multi-DNN mappings on NVIDIA AGX Xavier board. Specifically, we: **(i)** randomly partition each DNN to create sub-DNNs, **(ii)** randomly select a computing component (e.g., CPU, GPU, DLAs) for each sub-DNN, and **(iii)** randomly set an operating frequency for each computing component. This compiles a space of solutions (referred to as the design space), where we observe 4 classes of mappings: **(i)** mappings with both low throughput and high power consumption (labeled as **A**); **(ii)** mappings with high throughput but high power consumption (labeled as **B**); **(iii)** mappings with low throughput but low power

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676724

consumption (labeled as **C**); and **(iv)** mappings with high throughput and low power consumption (labeled as **D**). To classify for high and low throughput and high and low power consumption, we consider each space's extrema and quantize it into 16 bins. *We observe that only 3% of the solutions yield both high throughput and low power consumption. Hence, creating a multi-DNN manager to find optimal partition points given a multi-DNN workload and the corresponding computing components while also accounting for low power consumption is very complex due to the vast design space.*

In this work, we present **MapFormer**, a highly efficient transformer-based multi-DNN mapping manager for heterogeneous embedded devices. MapFormer performs fine-grained layer-splitting and distributes the workload across all the available computing components in order to boost system throughput and reduce power consumption. Our framework achieved 90.8% higher average inferences per Watt compared to the current state-of-art.

Overall, our main contributions are: ❶ We propose MapFormer, a highly accurate multi-DNN manager that utilizes fine-grain layer partitioning to increase system throughput and minimize power consumption. ❷ We utilize latent action pruning to boost our manager's convergence and accelerate its response time. ❸ To the best of our knowledge, we are the first to support DNN layer splitting across both the CPU, the GPU, and the DLAs.

Table 1: Comparison of state-of-art and MapFormer.

	MOSAIC [8]	ODMDEF [17]	GA [12]	OmniBoost [13]	**MapFormer**
Throughput optimization	✓	✓	✓	✓	✔
Power efficiency	✓	-	✓	-	✔
Multi-DNN workloads	-	-	✓	✓	✔
Deep Learning Accelerators Support	-	-	-	-	✔
Low Response Time	✓	✓	-	✓	✔

2 RELATED WORK

Authors in [15] propose a reinforcement learning-based (RL) framework for power and throughput optimization. However, their study is limited to ARM big.LITTLE MPSoCs. Authors in [2] propose the OD-RL method, which considers power and performance optimization on many-core systems. Still, they do not consider concurrent multi-DNN workload execution and specialized deep learning accelerators. Another work for efficient resource management of multicore systems was proposed in [7]. Nonetheless, they consider an FPGA board for their experimental case study, thus neglecting heterogeneous embedded systems. Similarly, authors in [18] propose CARTAD, an RL-based framework that utilizes DVFS on multicores for efficient scheduling. However, they target thermal optimization rather than power and throughput co-optimization. Regarding power-efficient DNN management on heterogeneous embedded devices, the authors in [1] propose the ARM-CO-UP framework, which increases throughput via sub-DNN pipelining for consecutive input frames. However, they do not consider concurrent execution of different DNNs. Similarly, the authors in [28] employ a wide set of computing components that operate on different precisions. However, their heuristic does not apply to conventional embedded devices. Another framework that targets power-constrained platforms is MOSAIC [8], which partitions DNNs to optimize throughput and power consumption. However, for the case of multi-DNN workloads, their framework does not consider other scheduled tasks, thus overloading the embedded GPU. ODMDEF [17] is another method that optimizes DNNs at the edge via a composite linear regression and k-NN cost model. However, they do not consider power consumption and only optimize for throughput. The authors in [12] utilize a genetic algorithm to distribute DNNs amongst the given computing components. However, this method employs unstable operators such as mutation, thus exhibiting slow convergence and requiring retraining for every distinct workload. OmniBoost [13] is the first framework to utilize a neural network as a cost model. However, OmniBoost does not leverage the available neural processing unit nor considers power consumption as a point of optimization. A qualitative comparison is presented in Table 1.

In summary, none of these works target multi-DNN throughput and power co-optimization or efficiently capitalize on the underlying heterogeneity via DL accelerator utilization.

3 PROBLEM FORMULATION

In this work, we target modern embedded systems that incorporate heterogeneity in terms of embedded GPUs and optimized deep learning accelerators (DLAs). MapFormer takes as input **(i)** a set of DNNs to be executed simultaneously, **(ii)** a collection of computing components denoted as C (e.g., CPUs, GPUs, DLAs), **(iii)** and a list of operational frequencies $\mathcal{F}^c$ for each component c in C. MapFormer searches the solution space for the mapping with the best trade-off between average throughput and overall power consumption while satisfying specific power constraints. MapFormer splits each DNN into smaller sub-DNNs to introduce fine-grained control over workload distribution. These sub-DNNs differ in architectural features and, as a result, have varying computational demands.

For each DNN in a multi-DNN workload find:
- **a layer splitting (i.e., sub-DNNs);**
- **a sub-DNN to computing component mapping; and**
- **a frequency for each computing component**

in order to:

❶ Maximize average workload throughput $\mathcal{T}$
❷ Minimize overall power consumption $\mathcal{P}$
such that
$\mathcal{P} \leq \mathcal{P}^{threshold}$

4 PROPOSED FRAMEWORK

Figure 2 presents a high-level overview of the proposed method. MapFormer consists of two primary components: **(i)** an estimator, which is a transformer-based classifier [6] that evaluates a mapping in terms of throughput and power consumption; and **(ii)** a Latent Action Monte Carlo Tree Search (LA-MCTS) [33] module that explores the extensive design space efficiently, utilizing tree pruning techniques within a predefined computational budget.

4.1 Input Formulation

As mentioned before, MapFormer takes as input: **(i)** a set of DNNs to be executed simultaneously; **(ii)** the set of available computing components; and **(iii)** the supporting operational frequencies for each one of these components. In this section, we detail how the input sequence S is structured to represent the mapping of

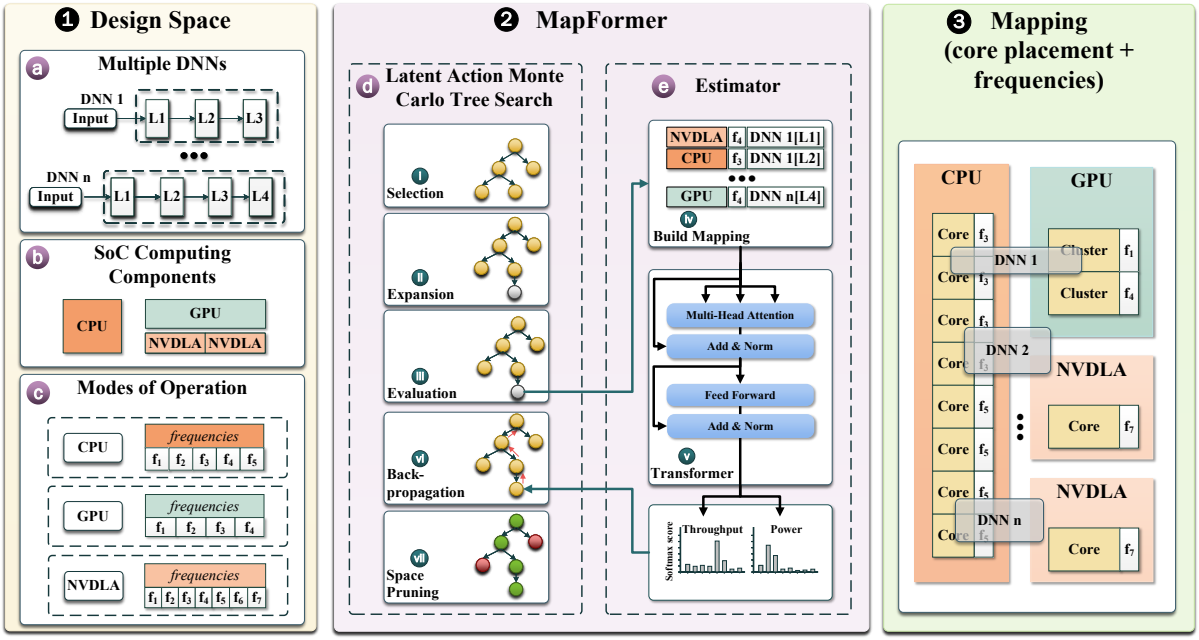

Figure 2: A high-level overview of MapFormer.

DNN layers to specific cores, along with frequency settings for each component. We achieve this through a composite embedding module [22] that incorporates: **(i)** the computational profile of each DNN layer within the workload, **(ii)** the processing capabilities of each computing component of the embedded device, and **(iii)** the operational frequency of all computing components. The pseudo-code for our highly scalable embedding module is given in listing 1. MapFormer utilizes layer partitioning to break down any DNN model into smaller sub-DNNs, requiring a layer-level input representation. To that end, for each layer in the workload, we apply our customized embedding module to create a sequence of tuples, each consisting of a layer, a computing component, and its corresponding operational frequency. Unlike previous methods [13, 17], our distributed embedding vectors are learnable, enhancing the transformer's ability to estimate throughput and power consumption more accurately. Transformers typically do not inherently understand tokens' relative or absolute positions in a sequence, so we incorporate a sinusoidal positional encoding layer [30].

```
class Embedding(nn.Module):
  def __init__(self, n_layers, n_ccomps, n_ops, size):
    self.l_emb = nn.Embedding(n_layers, size)
    self.cc_emb = nn.Embedding(n_ccomps, size)
    self.op_emb = nn.Embedding(n_ops, size)
    self.pos_emb = PositionalEncoding(size)

  def forward(self, l, cc, op):
    l_repr = self.l_emb(l) + self.pos_emb(l)
    cc_repr = self.cc_emb(cc) + self.pos_emb(cc)
    op_repr = self.op_emb(op) + self.pos_emb(op)
    return (l_repr, cc_repr, op_repr)
```

Listing 1: Composite embedding module.

4.2 Estimator Module

Building on the structure of our input sequence $\mathcal{S}$, we use a casual transformer-based estimator [32] to assess any mapping $\mathcal{M}$ and predict its throughput and power consumption. The choice of a transformer-based estimator is due to its ability to capture long-term dependencies, which is crucial for managing higher-order multi-DNN workloads—specifically, workloads where DNNs have a total of more than 1,000 layers to be mapped (e.g., 10 concurrent DNNs executed on the same board as shown in Section 5). Estimators from previous studies [12, 13, 17], although effective for smaller workloads, tend to underperform with larger multi-DNN workloads, often resulting in sub-optimal mappings. Our estimator is specifically trained to predict throughput and power. By focusing solely on sequence analysis rather than generation, our estimator can incorporate context from both directions at once, significantly enhancing the accuracy of predictions for both throughput and power consumption.

A major differentiator of MapFormer from previous state-of-the-art approaches is that it is designed for a classification task rather than a regression task [23]. Specifically, our transformer-based estimator predicts a distribution of scores for throughput and power consumption. While estimating exact values for these metrics could potentially yield better multi-DNN mappings, it also requires significantly larger datasets to manage the imbalances in target values [31]. For instance, mappings that achieve high throughput scores are relatively rare compared to those with lower throughput, creating an imbalance in the dataset that can lead to inaccurate predictions. To address this, we define the estimator's target as a distribution of N discrete classes, effectively transforming the problem into a classification task. We divide the value space into N equal bins in terms of sample size, which helps overcome data imbalance. Furthermore, to manage the multi-objective nature [11] of predicting both throughput and power consumption, we feed the contextualized sequence outputs from the transformer encoder

Table 2: MapFormer estimator VS casual encoder-only transformer. We deployed our estimator and the casual encoder-only transformer on NVIDIA AGX Xavier GPU to measure the achieved number of inferences per second.

Module	MapFormer	Casual	Comparison
QKV Heads	8	16	$\downarrow \times 2$
Atten. Layers	4	24	$\downarrow \times 6$
Embedding Dim.	64	1024	$\downarrow \times 16$
Feed-forward Size	256	4096	$\downarrow \times 16$
Parameters	$270K$	$303M$	$\downarrow \times 1,126$
Inf/sec	42	3	$\uparrow \times 14$

into two separate fully connected layers, each with N neurons corresponding to the classes in our target distribution.

Finally, to reduce the computational demands of our estimator, we use a lower-dimensional latent representation in our custom embedding layer, along with fewer multi-head attention layers and attention heads. Table 2 shows a detailed comparison of our transformer-based estimator to a standard encoder-only transformer, such as BERT [6] in terms of computational workload. The smaller parameter span also enables our estimator to converge fast and with a small dataset, rendering MapFormer highly efficient.

4.3 LA-MCTS Module

Our estimator module primarily works as a mechanism for mapping evaluation. To enhance this, we integrate a highly efficient space exploration module, the Latent Action-MCTS (LA-MCTS) [33]. MCTS is a heuristic approach that efficiently navigates extensive design spaces by iteratively interacting with its decision tree within a set computational budget [29]. This tree holds all possible mappings for a given design space. Although traditional MCTS effectively minimizes a cost function through stochastic processes, it tends to converge slowly. This slow convergence increases both the computational workload and the number of required estimator inferences.

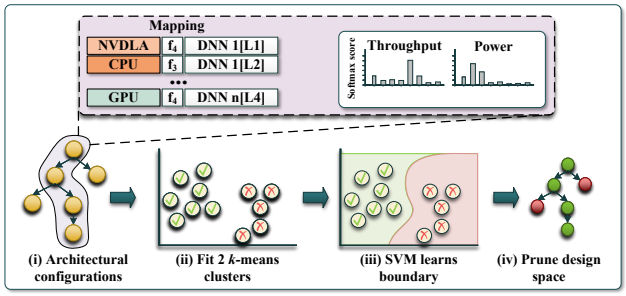

Figure 3: Design space pruning via LA-MCTS [33].

To enhance the convergence rate of MCTS, we adopted LA-MCTS, which iteratively learns to partition the design space hierarchically. In each iteration, LA-MCTS examines specific regions of the decision tree and applies a k-means algorithm to categorize them into two clusters, distinguishing between promising (good) and less promising (bad) solutions. It uses Support Vector Machines (SVM) [26] to create a decision boundary that extrapolates the patterns identified by the k-means to the broader design space. This process helps to prioritize the regions of the design space by assigning a likelihood score to each potential mapping, indicating its potential for further consideration. Figure 3 provides a high-level overview of the iterative process of LA-MCTS and how it prunes the design space to focus on more viable solutions.

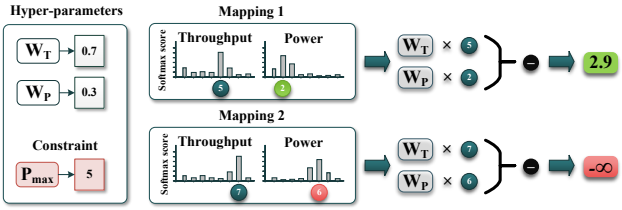

Figure 4: Function value for LA-MCTS.

To address the multi-objective nature of our problem, we formulate a composite value function $\mathcal{V}$. This function evaluates any mapping $\mathcal{M}$ by calculating the weighted difference between the predicted throughput class and the predicted power consumption class. Additionally, to ensure the satisfaction of the power constraint ($\mathcal{P}^{threshold}$), we incorporate a filtering step based on the predicted power consumption class. Specifically, if a mapping is projected to exceed the maximum allowable power consumption, it is assigned an infinitely negative value, effectively removing it from consideration as a viable solution. This approach is detailed mathematically in Equation 1.

$$\mathbf{V}(\mathcal{M}) = \begin{cases} \mathbf{W}_T \cdot \mathcal{T}(\mathcal{M}) - w_2 \cdot \mathbf{W}_P \cdot \mathcal{P}(\mathcal{M}) & \text{if } \mathcal{P}(\mathcal{M}) \leq \mathcal{P}^{threshold} \\ -\infty & \text{otherwise} \end{cases} \tag{1}$$

Here, $\mathbf{W}_T$ represents the weight assigned to throughput, $\mathcal{T}(\mathcal{M})$ denotes the estimated throughput class for the mapping $\mathcal{M}$, $\mathbf{W}_P$ is the weight assigned to power consumption, and $\mathcal{P}(\mathcal{M})$ indicates the predicted power consumption. The user-defined maximum allowable power consumption is denoted by $\mathcal{P}^{threshold}$. Figure 4 shows an example with two mappings. In this example, the first mapping has an estimated power consumption of 2, which is below the user-defined limit (5W), allowing its value to be computed and integrated back into the LA-MCTS tree. Conversely, the second mapping exceeds the power limit (5W) and is consequently assigned an infinite negative value, excluding it from viable solutions.

5 EXPERIMENTAL EVALUATION

In this section, we assess MapFormer's performance on throughput and power consumption trade-off efficiency by performing an in-depth evaluation of various multi-DNN workloads on NVIDIA Jetson AGX Xavier (JAX) [24]. The NVIDIA JAX features: (**i**) a Volta GPU [3] with 512 CUDA cores and 64 Tensor cores rendering a performance peak of 10 TFLOPS; (**ii**) a Carmel CPU with $\times 4$ ARMv8.2 dual-core clusters operating at $2.26GHz$; (**iii**) $\times 2$ NVIDIA Deep Learning Accelerators (NVDLAs); and (**iv**) a $32GB$ LPDDR4x memory. One core difference between the state-of-the-art and MapFormer lies in the capitalization of the NVDLAs. This is a crucial

feature of this work since NVDLAs are highly optimized computing components for DNN tasks. Specifically, some of the core NVDLA features are: (**i**) Convolution Core-optimized high-performance convolution engine; (**ii**) Single-point lookup engine for activation functions; (**iii**) Planar averaging engine for pooling; (**iv**) Multi-channel averaging engine for advanced normalization functions; (**v**) Memory-to-memory transformation acceleration for tensor reshape and copy operations; and (**vi**) Accelerated path to move data between two non-connected memory systems. The last feature ensures fast data transfer between the NVDLAs and the SoC GPU.

MapFormer is developed using the PyTorch framework [25]. To manage the multi-DNN workload mappings on NVIDIA JAX, we developed a PyTorch-powered compute library that supports fine-grained DNN partitioning. For the training phase of our transformer-based estimator, we generated 2,000 diverse workloads, each comprising random combinations of 1 to 10 DNNs, operating at various frequencies of computing components. Our PyTorch-based compute library enabled us to port all the models available under `torchvision.models`, creating a space of 80 widely used DNNs. These models are categorized into several families: (**i**) AlexNet; (**ii**) ConvNeXt; (**iii**) DenseNet; (**iv**) EfficientNet; (**v**) GoogLeNet; (**vi**) InceptionV3; (**vii**) MNASNet; (**viii**) MaxVit; (**ix**) MobileNetV2; (**x**) MobileNetV3; (**xi**) RegNet; (**xii**) ResNet; (**xiii**) ShuffleNetV2; (**xiv**) SqueezeNet; (**xv**) SwinTransformer; (**xvi**) VGG; and (**xvii**) VisionTransformer. MapFormer is designed to be compatible with any model defined in PyTorch. Each workload combination was randomly allocated among the computing components of the board. Our dataset generator assigned varying operational frequencies to the CPU clusters, the GPU, and the two NVDLAs [36], drawing from pools of 30 CPU frequencies, 15 GPU frequencies, and 4 NVDLA frequencies. Additionally, each frequency pool includes an "OFF" option, which allows our system to turn off a selected computing component. This feature facilitates more precise Dynamic Voltage and Frequency Scaling (DVFS) control over the board, enhancing power management.

Additionally, in the following experiments, we employed the following methods to evaluate MapFormer against: (**i**) **MOSAIC** [8], a linear regression approach; (**ii**) **ODMDEF** [17], a manager with both a linear regression and a k-NN classifier in its core; (**iii**) **GA**, the evolutionary manager proposed in [12]; and (**iv**) **OmniBoost** [13], a framework for greedy throughput optimization of multi-DNN workloads. We also add another variation of OmniBoost, namely **OmniBoost***, which involves retraining the CNN estimator that powers OmniBoost under different power thresholds.

5.1 Model Accuracy Evaluation

Current state-of-art DNN managers for heterogeneous embedded systems, such as MOSAIC, ODMDEF, GA, and OmniBoost, heavily depend on the accuracy of their cost models. To that end, we aim to showcase the efficiency of our estimator module compared to the state-of-art. We trained our estimator to classify into 16 throughput classes and 16 power classes. We chose these numbers to achieve an optimal balance between model accuracy and exploitation. Here, *accuracy* refers to the model's ability to correctly predict the target throughput and power distributions, while *exploitation* concerns the level of detail within those distributions. A higher number of classes in the distribution allows for greater detail, enhancing the effectiveness of the exploration mechanism, in this case, LA-MCTS. However, increasing the number of classes requires expanding the dataset, which adds complexity to the estimator's training. Therefore, to enable few-shot learning [35] and quickly capture the underlying data patterns, we found that a target distribution of 16 classes is enough to yield highly efficient multi-DNN mappings.

Table 3: Comaprison of estimator models regarding prediction accuracy across: (i) MOSAIC [8]; (ii) ODMDEF [17]; (iii) OmniBoost [13]; and (iv) MapFormer.

Cost Model	Throughput	Power Consumption
MOSAIC	46.5 ↓ 42.7%	55.8% ↓ 36.5%
ODMDEF	49.7% ↓ 39.5%	Not Appl.
OmniBoost	72.4% ↓ 16.8%	Not Appl.
MapFormer	89.2%	92.3%

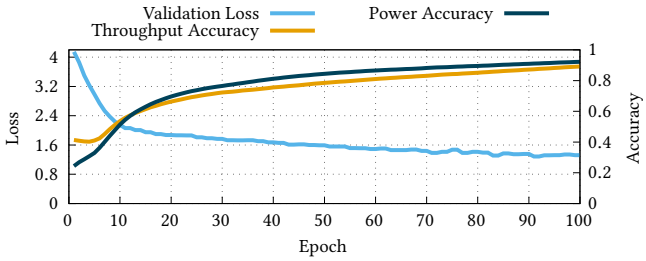

Figure 5: MapFormer's estimator training progress.

The estimator's accuracy strongly affects the quality of multi-DNN mappings that our MapFormer yields. Hence, we aim to boost its accuracy as much as possible. To that end, we trained our estimator for 100 epochs with 80% of our dataset using `AdamW` optimizer [21] with `0.0001` learning rate and `CosineAnnealingLR` scheduler [20] for smooth approximation of the global minimum cost during training. Figure 5 depicts our estimator's training progress regarding cross-entropy loss, throughput prediction accuracy, and power consumption prediction accuracy. Our estimator has successfully converged to a satisfactory accuracy for both tasks, i.e., throughput and power consumption prediction. To validate our claim, we evaluate our estimator on the remaining and unseen test subset. We also evaluate the prediction scores for Mosaic, ODMDEF, and OmniBoost. Table 3 lists the prediction capability of these frameworks compared to our MapFormer. MOSAIC is powered by a linear regression model, and thus, its evaluation score is the coefficient of determination of the prediction. ODMDEF is powered by one linear regression model and a k-NN [14], which is evaluated based on the mean accuracy of the given test data and labels. OmniBoost utilizes a CNN model, which solves a regression task of the throughput value function, similar to MOSAIC. Finally, ODMDEF and OmniBoost do not provide cost models for power consumption. It is evident that standard cost models, such as linear regression and k-NN, are rendered unreliable. These cost estimators fail to capture the underlying data patterns due to the dataset's complexity, necessitating cost models with higher learning capacity. This is verified with OmniBoost's performance, which is nonetheless worse by a

significant 16.8% compared to our MapFormer. The GA [12] cannot be evaluated in terms of accuracy since it is an unsupervised learning methodology. However, we demonstrate in Section 5.3 that the GA actually yields 28% worse throughput/power trade-off on average compared to OmniBoost, indicating a similar behavior to that of the ODMDEF.

There are two main reasons that negatively affect state-of-art cost models and lead to the prediction scores presented in Table 3. ❶ The number of DNNs in the given workload has been greatly scaled up. Specifically, the only framework that has attempted workloads of up to 5 concurrently executed DNNs is OmniBoost, whereas the rest of the multi-DNN managers have addressed workloads of at most 4 concurrently executed DNNs. **MapFormer** addresses the increased complexity of the problem by employing a transformer-based cost model capable of capturing data dependencies across long sequences and by utilizing distributional learning via cross-entropy criterion. This enables our framework to yield highly efficient mappings for workloads of up to 10 concurrently executed DNNs. ❷ The pool of DNN architectures considered in our work is greatly increased. Specifically, authors in OmniBoost experimented with a pool of 11 widely used DNNs, while the MOSAIC, the ODMDEF, and the GA consider smaller pools of less than 10 DNNs. **MapFormer** expands the valuation pool to 80 DNNs, thus increasing the complexity of the design space in terms of DNN computational profile diversity. To address that challenge, our cost model capitalizes on a 64-dimensional embedding layer that effectively captures the intricate properties of the DNN layers regarding both their computational profile and power requirements.

*Overall, state-of-art cost models heavily underperform in the problem's high-dimensional design space, thus resulting in inefficient multi-DNN mappings. We expand our analysis in Section 5.3 and further demonstrate **MapFormer**'s efficiency in finding multi-DNN mappings with high throughput and low power consumption.*

5.2 Value Function Evaluation

After establishing an accurate and efficient cost model, we focus on evaluating the value function used in the LA-MCTS module of MapFormer, as shown earlier in Figure 4. LA-MCTS generates a latent space and prunes less promising candidate solutions based on feedback from a return signal. Therefore, it is important to thoroughly analyze and fine-tune the hyperparameters associated with this return signal for optimal performance. To that end, we perform an in-depth study of the two hyper-parameters of our composite value function: (**i**) the weight of the estimated throughput level, $\mathbf{W}_T$; and (**ii**) the weight of the estimated power consumption level, $\mathbf{W}_P$. To explore these parameters efficiently, we generated a diverse set of multi-DNN mixes by pooling different DNNs. Specifically, we created 60 unique multi-DNN mixes, with 10 mixes each for configurations ranging from 5 to 10 concurrently executing DNNs. For each mix, we considered the following metrics:

❶ **Average throughput $\mathcal{T}$ (inferences per second):** The average throughput is defined as $\mathcal{T} = \frac{\sum_{i=1}^{N} t^i}{N}$, where $t^i = \frac{inferences}{seconds}$ represents the throughput in inferences per second for each DNN i in the mix.

❷ **Total power consumption $\mathcal{P}$ (W):** Defined as $\mathcal{P} = \mathbf{P}_{gpu} + \mathbf{P}_{cpu} + \mathbf{P}^1_{dla} + \mathbf{P}^2_{dla} + \mathbf{P}_{emc}$, accounting for the power consumed by all of the available computing components and the SoC memory.

❸ **Efficiency $\mathcal{E}$ (inferences per Watt):** To assess each multi-DNN manager in terms of throughput and power consumption trade-off, we define the efficiency metric as $\mathcal{E} = \frac{\sum_{i=1}^{N} t^i}{\mathcal{P}}$.

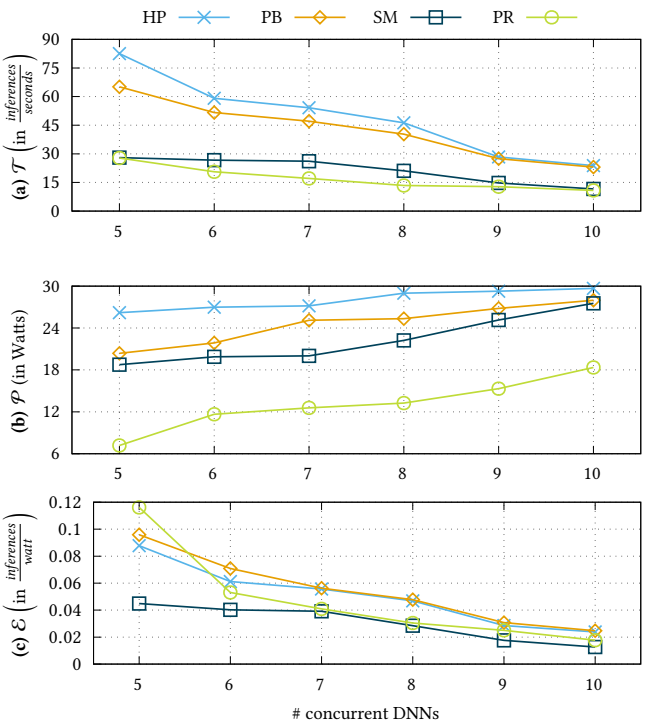

Figure 6: Comparison of MapFormer's: (i) High Performance (HP) mode; (ii) Performance Boost (PB) mode; (iii) Standard Mode (SM); and (iv) Power Reduce (PR) mode. We evaluate each mode in terms of: (a) Average throughput ($\mathcal{T}$); (b) Total power consumption ($\mathcal{P}$); and (c) Inferences per Watt ($\mathcal{E}$) across workloads of 5 up to 10 concurrently executed DNNs.

We identify 4 modes of operation for our MapFormer: (**i**) High Performance (HP) mode ($\mathbf{W}_T = 1.0, \mathbf{W}_P = 0.0$), which considers only the user-defined power constraint as its only objective regarding power consumption and shifts its focus heavily on high throughput; (**ii**) Performance Boost (PB) mode ($\mathbf{W}_T = 0.7, \mathbf{W}_P = 0.3$), which prioritizes average throughput over total power consumed; (**iii**) Standard Mode (SM) ($\mathbf{W}_T = 0.5, \mathbf{W}_P = 0.5$), which attempts to find the best balance between throughput and power consumption; and (**iv**) Power Reduce (PR) mode ($\mathbf{W}_T = 0.3, \mathbf{W}_P = 0.7$), which strives to keep lower power consumption while sacrificing some performance regarding average workload throughput. Finally, we do not consider any power constraint in this part, i.e., the power consumption threshold level is set equal to the maximum operational power of the board.

Figure 6(a) shows that both HP and PB modes achieve high average throughput $\mathcal{T}$. All methods gradually fall in terms of $\mathcal{T}$ as we progressively stress the board with more concurrent DNNs to be executed. However, both HP and PB modes surpass SM and PR

modes regarding mixes of 9 and 10 concurrently executing DNNs with almost double $\mathcal{T}$.

Figure 6(b) complements our analysis by analyzing the behavior of MapFormer's modes regarding total power consumption $\mathcal{P}$. It is evident that while PR mode did not perform well regarding $\mathcal{T}$, it achieves 63.9%, 47.8%, and 42.2% lower power consumption compared to HP, PB, and SM, respectively. This strong correlation between high throughput $\mathcal{T}$ and HP mode of operation, as well as total power consumption $\mathcal{P}$ and PR strengthens our claim regarding the accuracy and prediction skill of our estimator module and provides valuable insight regarding the behavior of our value function. Finally, HP is almost a straight line around 30 Watts, which is the board maximum power consumption setting, since it does not consider power consumption minimization as an objective and shifts all the weight to maximizing the average throughput $\mathcal{T}$.

Finally, Figure 6(c) depicts the efficiency $\mathcal{E}$ of each mode regarding throughput and power consumption trade-off. We observe that for the case of 5 DNNs in the workload, PR mode achieves the best throughput/power consumption trade-off. This is because the board is not yet stressed, so there is a large margin for optimization regarding power consumption, which MapFormer finds while operating in PR mode. However, this efficiency is inconsistent as the computational demands increase and more DNN tasks are considered in the workload. This results in PB and HP surpassing PR regarding $\mathcal{E}$. All modes score close $\mathcal{E}$ since there is a lot of power consumed by default under such heavy order multi-DNN workloads. Finally, SM yields the smallest fluctuations, rendering almost a straight line regarding $\mathcal{E}$, proving its ability to efficiently balance both throughput and power consumption.

Overall, PB mode demonstrates the highest $\mathcal{E}$ on average out of all 4 candidate modes of operation, and hence we will consider $\mathcal{W}_T$ = 0.7 and $\mathcal{W}_P$ = 0.3 for our MapFormer regarding the rest of our experimental evaluation.

5.3 Mapping Efficiency Evaluation

In this section, we demonstrate the efficiency of our framework against 4 other state-of-art managers, namely the MOSAIC, the ODMDEF, the GA, OmniBoost, and OmniBoost*. Specifically, we leverage the `tegrastats` tool of NVIDIA JAX [9] and build groups of data samples under the same power mode, thus aiding OmniBoost to realize the space of multi-DNN workload power consumption. We consider 3 distinct multi-DNN workload scenarios, each characterized by a different level of computational complexity: (**a**) A computationally lightweight workload, comprising 6 concurrently executing DNNs, which represents a scenario with reduced computational demands; (**b**) A moderate computational workload, consisting of 8 concurrently executing DNNs, exemplifying a balanced computational requirements; and (**c**) A computationally demanding workload, involving 10 concurrently executing DNNs, embodying a scenario with increased computational complexity.

To evaluate each manager in terms of power consumption, we define three distinct levels of operation for NVIDIA JAX. The first level of operation is a heavily constrained power mode, restricting the combined frequencies of the computing components to a total power consumption of 10 Watts. We assess the ability of each manager to account for the changes in the computing performance of the platform's computing components while operating

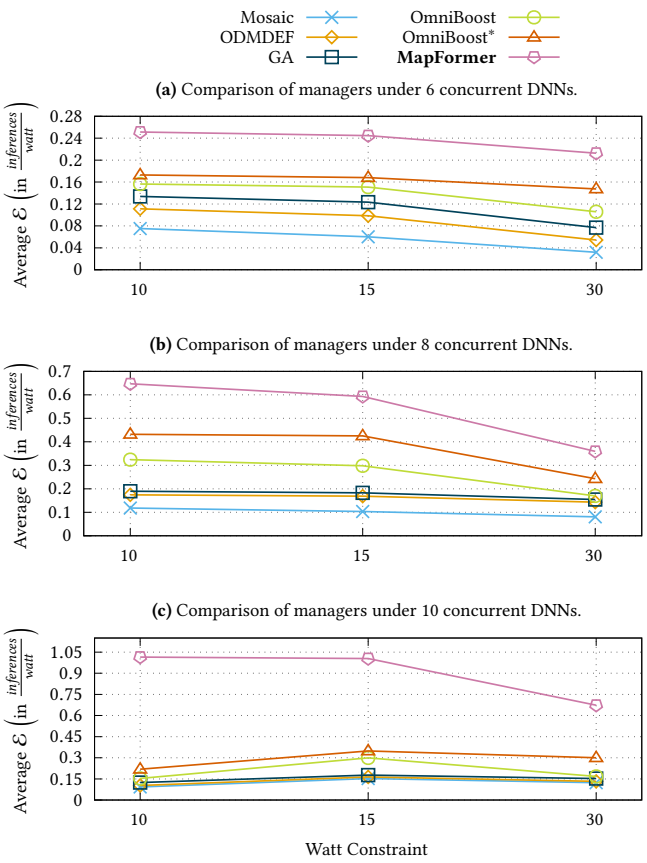

Figure 7: Comparison of Inferences per Watt $\mathcal{E}$ between MapFormer and SOTA considering (a) low computational workload mixes of 6 concurrent DNNs; (b) medium computational workload mixes of 8 concurrent DNNs; and (c) high computational workload mixes of 10 concurrent DNNs.

at lower frequencies. The second mode relaxes the power consumption constraints, allowing for frequencies that result in a total power consumption of 15 Watts. This mode enables the evaluation of each manager's capability to leverage the increased computing resources to achieve higher average workload throughput without significantly compromising power efficiency. Lastly, the third mode assesses the performance of each manager in the absence of any strict user-defined power constraints, focusing on their ability to map multi-DNN workloads in order to boost average throughput under unconstrained power conditions. Importantly, the maximum power budget of the board is 30 Watts. For each mode, we pool different DNNs, thus creating 30 unique multi-DNN mixes, with 10 mixes for each mode. Finally, we average the efficiency $\mathcal{E}$ achieved by each manager across these mixes. This comprehensive evaluation methodology includes a range of power consumption scenarios and provides valuable insights into the robustness and efficiency of each manager in handling different operational requirements.

Figure 7(a) depicts the performance of state-of-art managers compared to **MapFormer** for the computationally lightweight scenario of 6 concurrently executing DNNs. MapFormer demonstrates

higher efficiency $\mathcal{E}$ achieving an average improvement of 76.4%, 62.7%, 52.8%, 41.7%, and 31.1% over MOSAIC, ODMDEF, GA, OmniBoost, and OmniBoost*, respectively. This significant performance difference is due to a core feature of MapFormer; to the best of our knowledge, MapFormer is the first framework to support layer splitting across NVDLAs together with the platform's CPU and GPU. By effectively leveraging the diverse computational capabilities of these components, MapFormer optimizes the execution of multi-DNN workloads, resulting in enhanced efficiency even under reduced computational demands.

MapFormer performs consistently under moderate computational workloads consisting of 8 concurrently executing DNNs, as illustrated in Figure 7(b). It maintains its lead regarding mapping efficiency $\mathcal{E}$, surpassing by 81.1%, 69.6%, 67.1%, 50.4%, and 31.2% the MOSAIC, ODMDEF, GA, OmniBoost, and OmniBoost*, respectively. Notably, MOSAIC, ODMDEF, and GA exhibit limited scalability with respect to the power constraint, resulting in nearly constant $\mathcal{E}$. This observation underscores the necessity for more sophisticated cost models to effectively manage multi-DNN workloads under varying power constraints.

Figure 7(c) showcases the performance comparison under the most computationally demanding scenario, involving 10 concurrently executing DNNs. The results highlight MapFormer's significant contribution to multi-DNN workload management, effectively co-optimizing throughput and power consumption. MapFormer shows robustness in mapping efficiency $\mathcal{E}$ under the heavily constrained 10 Watt mode, surpassing MOSAIC, ODMDEF, GA, OmniBoost, and OmniBoost* by 90.8%, 89.9%, 87.8%, 84.9%, and 78.4%, respectively. This better performance is due to MapFormer's highly accurate throughput and power consumption estimator, which leverages a transformer-based architecture to correlate computing components, frequencies, and DNN layers in a high-dimensional space via its composite embedding layer. In contrast, current run-time multi-DNN managers struggle to capture long-term dependencies in higher-order multi-DNN workloads due to their less capable cost models. Even under the unconstrained power scenario of 30 Watts, MapFormer significantly outperforms other managers. This is due to the competing managers' underutilization of the underlying hardware heterogeneity. MapFormer, on the other hand, fully capitalizes on this heterogeneity by distributing layers across the NVDLAs in addition to the CPU and GPU. This efficient utilization of the platform's computational resources enables MapFormer to achieve superior performance and power efficiency across a wide range of multi-DNN workloads and power constraints.

In summary, the experimental evaluation demonstrates MapFormer's exceptional efficiency and robustness in managing multi-DNN workloads, consistently outperforming state-of-the-art managers across various computational complexities and power constraints.

5.4 Run-time Performance Evaluation

In this section, we evaluate the execution time of each manager. MOSAIC and ODMDEF exhibit poor scalability with respect to workload complexity, as they create a query for their cost models for each DNN layer in the workload, resulting in a complexity of $O(\mathrm{N})$, where N is the number of layers to be mapped. Despite their lightweight cost models, they achieve response times of $\sim$ 1 sec. and $\sim$ 3 sec., respectively. However, as demonstrated in Section 7, both managers find inefficient mappings in terms of throughput and power consumption. Among the current state-of-the-art managers, OmniBoost ranks second with $O(\mathbf{1})$ complexity, utilizing also MCTS with a constant exploration budget. However, to accommodate the increased complexity of our problem, which considers $\times 2$ more concurrently executing DNNs in the workload compared to the authors' experimental evaluation, we had to significantly increase the number of iterations. Consequently, OmniBoost achieves a $\sim$ 60 second response time while rendering multi-DNN mappings that are, on average, 56.4% less efficient than MapFormer in terms of throughput and power consumption. The GA exhibits the worst complexity, as it does not employ any cost model and requires offloading every chromosome in the population for each generation to be evaluated in real-time by the NVIDIA JAX. This results in an average response time of $\sim$ 2 hours for each multi-DNN workload, considering a low population of 10 chromosomes and a similarly low number of generations (20). Regarding **MapFormer**, by leveraging advanced latent space pruning (depicted in Figure 3), we set the number of iterations to 100. Additionally, by utilizing flash attention V2 [4] in our estimator, we avoid the quadratic complexity associated with casual transformers. These optimizations result in an average response time of $\sim$ 23 seconds for MapFormer.

In summary, MapFormer achieves the best balance between run-time performance and efficient multi-DNN workload management across all scenarios, rendering it the most robust and effective framework among the evaluated approaches.

6 DISCUSSION & FUTURE WORK

In this paper, we presented **MapFormer**, a robust framework for multi-DNN management that effectively addresses diverse workload scenarios and platform power constraints. MapFormer tackles the high dimensionality of the design space by modeling the problem as a classification task through distributional learning. By leveraging the NVDLAs of the target system, MapFormer achieves a more efficient balance between throughput and power consumption than state-of-the-art solutions. Looking ahead, we plan to extend our framework to support knowledge transfer, enabling the seamless porting of pre-trained weights from our estimator to various embedded systems with little to no fine-tuning. This will greatly enhance the adaptability and scalability of MapFormer across different hardware platforms. Furthermore, we aim to explore cache utilization strategies to optimize throughput further and reduce power consumption. By increasing cache utilization through DNN operation and weight sharing, especially amongst DNNs in the same family (e.g., VGG-11 and VGG-16), we can minimize memory access latency and improve overall system performance. Despite the promising avenues for future research, MapFormer already presents a novel approach to throughput and power consumption co-optimization. To the best of our knowledge, it is the first framework capable of capitalizing on the board's deep learning accelerators together with its CPU and GPU, unlocking new possibilities for efficient multi-DNN workload management on embedded systems.

ACKNOWLEDGMENTS

This work is supported by grant NSF CCF 2324854.

REFERENCES

[1] Ehsan Aghapour, Dolly Sapra, Andy Pimentel, and Anuj Pathania. 2024. ARM-CO-UP: ARM CO operative U tilization of P rocessors. *ACM Transactions on Design Automation of Electronic Systems* (2024).

[2] Zhuo Chen, Dimitrios Stamoulis, and Diana Marculescu. 2017. Profit: priority and power/performance optimization for many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (2017), 2064–2075.

[3] Jack Choquette, Olivier Giroux, and Denis Foley. 2018. Volta: Performance and programmability. *Ieee Micro* 38, 2 (2018), 42–52.

[4] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[5] Abhijit Das, Enrico Russo, and Maurizio Palesi. 2024. Multi-Objective Hardware-Mapping Co-Optimisation for Multi-DNN Workloads on Chiplet-based Accelerators. *IEEE Trans. Comput.* (2024).

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

[7] Bryan Donyanavard, Tiago Mück, Amir M Rahmani, Nikil Dutt, Armin Sadighi, Florian Maurer, and Andreas Herkersdorf. 2019. Sosa: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 685–698.

[8] Myeonggyun Han et al. 2019. Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE.

[9] Stephan Holly, Alexander Wendt, and Martin Lechner. 2020. Profiling energy consumption of deep neural networks on nvidia jetson nano. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*. IEEE, 1–6.

[10] Chenying Hsieh, Ardalan Amiri Sani, and Nikil Dutt. 2019. Surf: Self-aware unified runtime framework for parallel programs on heterogeneous mobile architectures. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 136–141.

[11] Ronghang Hu and Amanpreet Singh. 2021. Unit: Multimodal multitask learning with a unified transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1439–1449.

[12] Duseok Kang et al. 2020. Scheduling of deep learning applications onto heterogeneous processors in an embedded device. *IEEE Access* (2020).

[13] Andreas Karatzas and Iraklis Anagnostopoulos. 2023. OmniBoost: Boosting Throughput of Heterogeneous Embedded Devices under Multi-DNN Workload. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE.

[14] Oliver Kramer and Oliver Kramer. 2013. K-nearest neighbors. *Dimensionality reduction with unsupervised nearest neighbors* (2013), 13–23.

[15] Eunji Kwon, Sodam Han, Yoonho Park, Jongho Yoon, and Seokhyeong Kang. 2021. Reinforcement learning-based power management policy for mobile device systems. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 10 (2021), 4156–4169.

[16] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. 2019. On-device neural net inference with mobile gpus. *arXiv preprint arXiv:1907.01989* (2019).

[17] Cheolsun Lim and Myungsun Kim. 2021. ODMDEF: on-device multi-DNN execution framework utilizing adaptive layer-allocation on general purpose cores and accelerators. *IEEE Access* 9 (2021), 85403–85417.

[18] Di Liu, Shi-Gui Yang, Zhenli He, Mingxiong Zhao, and Weichen Liu. 2021. CARTAD: Compiler-assisted reinforcement learning for thermal-aware task scheduling and dvfs on multicores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 6 (2021), 1813–1826.

[19] Google LLC. 2024. Google Lookout. https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.reveal&hl=en_US&gl=US. Accessed: 2024-05-05.

[20] Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

[21] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[22] Tomas Mikolov et al. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[23] Shinichi Morishita. 1998. On classification and regression. In *International Conference on Discovery Science*. Springer, 40–57.

[24] NVIDIA. 2024. NVIDIA Jetson Xavier Series. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/. Accessed: 2024-05-05.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[26] Arti Patle and Deepak Singh Chouhan. 2013. SVM kernel functions for classification. In *2013 International conference on advances in technology and engineering (ICATE)*. IEEE, 1–9.

[27] Semiengineering. 2023. ML Moves From Servers To Smart Phones. https://semiengineering.com/ml-moves-from-servers-to-smart-phones/. Accessed: 2024-05-05.

[28] Ourania Spantidi et al. 2022. Targeting DNN Inference via Efficient Utilization of Heterogeneous Precision DNN Accelerators. *IEEE Transactions on Emerging Topics in Computing* (2022).

[29] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review* 56, 3 (2023), 2497–2562.

[30] Sho Takase and Naoaki Okazaki. 2019. Positional encoding to control output sequence length. *arXiv preprint arXiv:1904.07418* (2019).

[31] Fadi Thabtah, Suhel Hammoud, Firuz Kamalov, and Amanda Gonsalves. 2020. Data imbalance in classification: Experimental evaluation. *Information Sciences* 513 (2020), 429–441.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[33] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. 2020. Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems* 33 (2020), 19511–19522.

[34] Yingchun Wang, Jingyi Wang, Weizhan Zhang, Yufeng Zhan, Song Guo, Qinghua Zheng, and Xuanyu Wang. 2022. A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks* 8, 1 (2022), 1–17.

[35] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[36] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. 2018. Research on NVIDIA deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. IEEE, 192–195.