Tango: Low Latency Multi-DNN Inference on Heterogeneous Edge Platforms

Zain Taufique
University of Turku
Turku, Finland
zatauf@utu.fi

Aman Vyas
University of Turku
Turku, Finland
amvyas@utu.fi

Antonio Miele
Politecnico di Milano
Milan, Italy
antonio.miele@polimi.it

Pasi Liljeberg
University of Turku
Turku, Finland
pasi.liljeberg@utu.fi

Anil Kanduri
University of Turku
Turku, Finland
spakan@utu.fi

***Abstract*—There is an increasing demand to run DNN applications on edge platforms for low-latency inference. Executing multi-DNN workloads with diverse compute and latency requirements on resource-constrained heterogeneous edge platforms poses a significant scheduling challenge. In this work, we present *Tango* framework for orchestrating multi-DNN inference on heterogeneous edge platforms. Our approach uses a Proximal Policy-based Reinforcement Learning agent to jointly optimize cluster selection, accuracy configuration, and frequency scaling to minimize inference latency with a tolerable accuracy loss. We implemented the proposed *Tango* framework as a portable middleware and deployed it on real hardware of the Jetson TX edge platform. Our evaluation against relevant multi-DNN scheduling strategies demonstrates 61% lower latency and 48.4% lower energy consumption at a maximum accuracy loss of 1.59%.**

***Index Terms*—Edge AI, DNN inference, Heterogeneous systems**

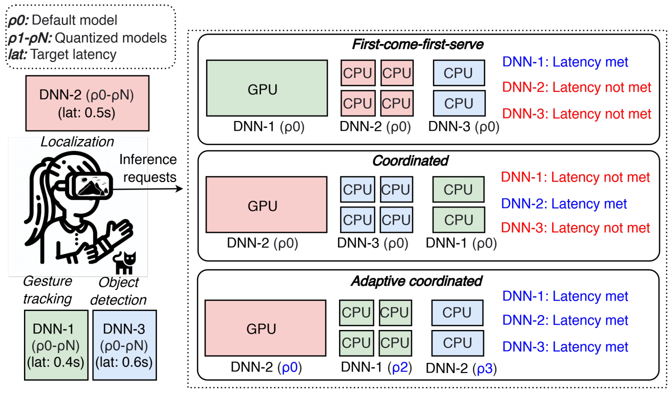

Fig. 1. Orchestrating concurrent DNNs with different scheduling strategies

I. Introduction

There is an increasing demand to run Deep Neural Network (DNN) inference on user-end mobile and edge devices to minimize latency and improve response time in applications such as autonomous vehicles, smart buildings, and augmented/virtual reality (AR/VR), etc [1]. Most applications require simultaneous and low-latency inference of multiple DNNs to deliver complex autonomous services. For example, autonomous vehicles use multiple DNNs for object detection and classification, while AR/VR applications use them for user tracking and pose estimation [2]. Under multi-DNN workloads, inference latency of a DNN is affected due to shared resource contention with other concurrently running DNNs. (i) Limited compute capacity and energy budgets, and core-level heterogeneity (e.g., asymmetric CPUs, GPUs, and custom accelerators) of edge devices, (ii) diversity in compute characteristics and latency requirements of concurrent DNNs, and (iii) stochastic run-time variance (e.g., cluster availability, workload variation) of running dynamic multi-DNN workloads exacerbate the multi-DNN inference challenge. Minimizing DNN inference latency within these constraints requires intelligent run-time orchestration to schedule multi-DNN workloads efficiently.

Existing multi-DNN inference scheduling techniques use offline profiling to partition DNNs into multiple layers and schedule each partition on a different cluster in a coordinated fashion [3], [4]. These approaches primarily focus on ensuring global throughput; they achieve low latency inference for the DNN mapped onto the GPU while compromising the latency of other concurrent DNNs mapped on the CPU. Other approaches [2], [5] enhanced coordinated DNN scheduling with Dynamic Voltage/Frequency Scaling (DVFS), although aggressive frequency scaling has only limited gains considering the stringent power budgets of edge devices. Existing multi-DNN inference techniques thus inevitably degrade the latency requirements of concurrent DNNs, necessitating alternative approaches for low latency inference.

From the applications' standpoint, we note that most of the DNNs are error resilient due to their algorithmic nature, input data quality, and results' format. Edge inference techniques exploit these DNN properties through model pruning, compression, and quantization for low latency inference [6]. We posit that opportunistically using pruned/quantized models jointly with appropriate cluster selection improves per-inference latency of multi-DNN workloads. When a DNN has to be inevitably mapped on a sub-optimal cluster under multi-DNN workload, this approach mitigates the latency penalty by trading off accuracy within a tolerable range.

Figure 1 presents a multi-DNN workload with different compute and inference latency requirements (0.4s, 0.5s, 0.6s respectively for DNNs 1-3) executed on a heterogeneous edge platform. In this example, the *First-come-first-serve (FCSFS)* scheduler maps the first arriving DNN-1 onto the GPU, and the subsequent DNN-2 and DNN-3 onto the CPU clusters, resulting in higher latency of DNN-2 and DNN-3. State-of-the-art coordinated scheduling approaches map each DNN to a different cluster based on offline profiling. In Figure 1, the coordinated scheduler maps DNN-2 onto the GPU, DNN-3 onto CPU cluster-1, and DNN-1 onto CPU cluster-2. This ensures low latency inference of DNN-2 onto the GPU; however DNN-1 and DNN-3 potentially fail to meet the latency requirements on CPU clusters. While coordinated scheduling makes optimal cluster selection decisions for standalone DNN requests, such approaches fail in providing latency guarantees under multi-DNN requests. It should be noted that both FCSFS and *Coordinated* scheduling approaches use the default DNN models ($\rho0$) without considering the usage of pruned/quantized models for lower latency. Finally, the *Adaptive coordinated* scheduler jointly (i) selects the appropriate cluster for executing each DNN and (ii) simultaneously configures the accuracy of the DNN kernels ($\rho2$ for DNN-1 and $\rho3$ for DNN-2) such that target inference latency of the DNN request is met on the selected cluster. Hence, an adaptive coordinated runtime resource management framework is required to leverage the configurable accuracy-latency trade-off, and the cluster heterogeneity to meet the intense workload requirements at a minimal accuracy loss. It should be highlighted that the cluster placement of *coordinated* and *adaptive coordinated* strategies can be different because selecting a quantized model changes the workload dynamics of the DNN kernels.

The example presented in Figure 1 is a simplified instance of multi-DNN workloads. In practical scenarios, implementing adaptive coordinated scheduling requires wider exploration of per-cluster latency-accuracy trade-offs, consideration of latency requirements across multi-DNN workloads, and interfaces for enforcing scheduling decisions. Given these motivations, we propose *Tango*, a multi-DNN inference framework for mapping concurrent DNNs requests on heterogeneous clusters while opportunistically using pruned/quantized models to meet latency requirements of each DNN inference. Our approach considers diverse compute and latency requirements of all the concurrently running DNNs to jointly (i) select an appropriate cluster for a given DNN, (ii) configure the accuracy of the DNN kernel at run-time such that target inference latency of the DNN request is met on the selected cluster, and (iii) fine tunes the performance by actuating per-cluster DVFS. When a suitable cluster is busy executing a workload, our approach addresses the relatively high latency of DNN inference requests scheduled on a lower-performing cluster through accuracy configuration. Under dynamic multi-DNN requests, our approach efficiently switches among different clusters and accuracy configurations to meet latency requirements of all the currently running DNNs. Considering the complexity of run-time joint cluster selection and accuracy configuration on heterogeneous edge devices, we formulate multi-DNN scheduling as a Reinforcement Learning (RL) problem. We design a Proximal Policy Optimization (PPO) based RL agent that makes joint cluster selection and accuracy configuration decisions to minimize inference latency at minimum output accuracy loss. PPO based RL agent is suitable for our problem statement because it provides the probability distribution of actions based on observations.
Our novel contributions include:

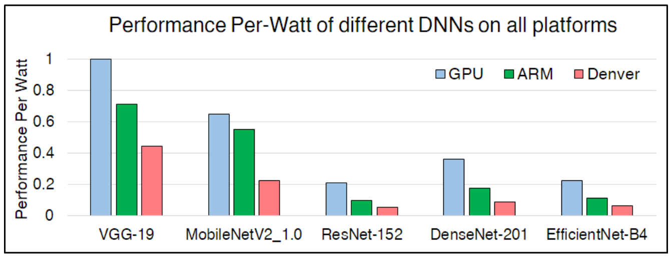

Fig. 2. Comparison of inference performance-per-watt of different DNNs.

- Multi-DNN inference framework that can handle concurrent DNN requests with varying compute and latency requirements and accuracy constraints on heterogeneous edge platforms
- Design of RL agent for joint cluster selection and accuracy configuration of multi-DNN workloads to minimize inference latency and maximize prediction accuracy
- Evaluation of the proposed solution on real hardware testbed of Jetson TX embedded platform with 17 different combinations of DNN kernels, achieving upto 61% improved latency and 48% lower energy consumption

The paper is organized as follows: Section II provides background and motivation for our proposed approach, Section III presents an overview of our framework infrastructure, Section III-B elaborates our proposed strategy. Section IV evaluates our proposed solution against other relevant strategies, followed by conclusions in Section V.

II. Background and Motivation

This section presents the challenges and opportunities for inferring multi-DNN workloads on heterogeneous platforms with CPU and GPU support. We used Jetson TX board [7], with a Pascal GPU, heterogeneous CPU clusters (quad-core ARM cluster and dual-core Denver cluster) as a baseline setup to demonstrate multi-DNN inference challenges.

A. Utilizing Device Heterogeneity

Modern edge platforms, supported by heterogeneous CPU and GPU clusters, exhibit diverse energy-latency characteristics for multi-DNN workloads. Figure 2 shows the performance-per-watt of inferring five different DNN kernels on the Jetson TX. The convolution layers in DNNs have the highest computation and energy requirement to perform numerous Multiply–Accumulate operation (MAC) operations [8]. GPU and ARM clusters report low inference latency of the DNNs because they allow parallelization of the convolution operations. In contrast, the Denver cluster shows the highest latency because it is inherently designed to perform single-threaded operations. GPU has the highest power consumption but reports the highest performance-per-watt due to significantly lower latency. The non-uniform energy-latency characteristics of heterogeneous devices expose large design space to schedule diverse multi-DNN workloads.

B. Latency penalty with multi-DNN workloads

Figure 3 shows the inference latency of five DNNs when executed in (i) standalone (only 1 DNN) and (ii) multi-DNN (executed simultaneously with other DNNs) modes

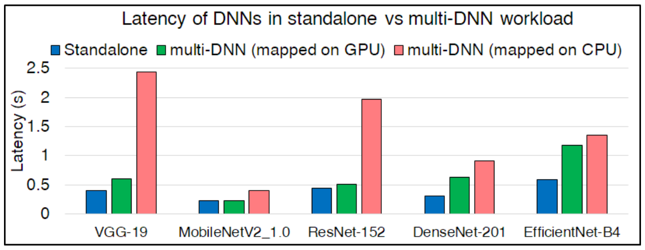

Fig. 3. Latency of DNN kernels in standalone vs multi-DNN workloads.

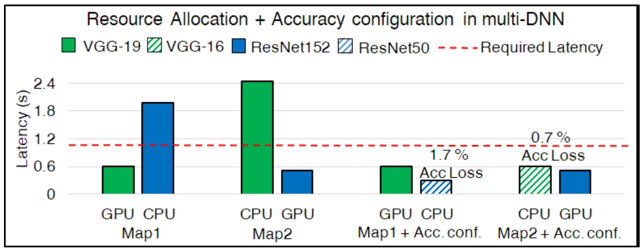

Fig. 4. Latency of VGG + ResNet combination with different cluster selection and accuracy configuration choices.

on the CPU and GPU clusters. For multi-DNN workloads, we used the combinations of VGG-19 + MobileNetV2_1.0, MobileNetV2_1.0 + ResNet-152, ResNet-152 + DenseNet-201, DenseNet-201 + EfficientNet-B4, and EfficientNet-B4 + VGG-19 respectively. The standard Linux governors prioritize GPU mapping of the DNN applications in the standalone mode by default. As shown in Figure 3, inference latency of each DNN increases when executed under multi-DNN workloads, while the latency penalty is significantly higher when run on the CPU clusters. For example, VGG-19 has an inference latency of 0.4s when executed on GPU in standalone mode. However, it increases to 0.6s when MobileNetV2 concurrently runs on the ARM cluster. Further, the inference latency of VGG-19 increases to 2.43s when executed on the ARM cluster with MobileNetV2 concurrently running on the GPU. It should be noted that in Figure 3, the latency penalty is different for different combinations of multi-DNN workloads, due to diverse compute characteristics of DNNs. Moreover, the perceived impact of the latency penalty is subject to the latency requirements of a given inference request. For example, a VGG-19 inference request with a target latency of 3s does not get affected under the multi-DNN combination of VGG-19 + MobileNetV2. Conversely, the same inference request with a target latency of 0.5s violates the latency requirement under a multi-DNN workload. Our proposed *Tango* framework handles the aforementioned challenges through run-time joint cluster selection and accuracy configuration by considering latency requirements of all the concurrently running DNNs.

C. Resource Allocation Example

Joint optimization of cluster selection and accuracy configuration can minimize the inference latency of multi-DNN workloads on sub-optimal clusters. Figure 4 shows a motivational example of inferring two concurrent DNN kernels including VGG and ResNet on Jetson TX where both the applications require a minimum latency of 1s. With the placement configuration Map1, VGG-19 is mapped on GPU, and Resnet152 is mapped on the ARM-CPU cluster. Here, VGG-19 takes 0.6s per inference at 73% output accuracy, while ResNet-152 is lagging the target with 1.8s latency at an output accuracy of 76.6%. At Map2, the CPU and GPU mapping is switched for VGG-19 and ResNet152. Now the VGG-19 takes 2.4s per inference, while the latency of ResNet152 is improved to 0.59s per inference. In these mapping configurations, all CPUs and GPU are running at the highest frequencies, and yet at least one DNN kernel is underperforming. The experiment shows that both cluster mapping and DVFS have computational and energy limitations, requiring an additional control knob is required to achieve the target latency for a multi-DNN workload. For Map1, we configure the accuracy of VGG-19 by switching the kernel to VGG-16, achieving 0.65s latency on the ARM-CPU cluster at an accuracy loss of 1.7%. Similarly for Map2, we configure the accuracy of ResNet152 by switching to ResNet50, achieving an inference latency of 0.65s at an accuracy loss of 0.7%. In this scenario, Map2 with accuracy configuration is a suitable option since it meets the target latency at a minimal accuracy loss. Our proposed *Tango* framework aims to search optimal combinations for low latency multi-DNN inference.

TABLE I
COMPARISON OF TANGO WITH STATE-OF-THE-ART RESEARCH WORKS

	Acc. aware	Platform	Multi -DNN	Strategy
pipeit [3]	✘	Asymmetric	✓	DNN pipelining
OmniBoost [4]	✘	heterogeneous	✓	DNN pipelining
MOC [9]	✘	heterogeneous	✘	Placement + DVFS
Band [2]	✘	heterogeneous	✓	Pipelining + DVFS
Kim et. al [10]	✘	heterogeneous	✓	DNN pipelining
Kang et. al [6]	✓	heterogeneous	✘	Quantized+pipelined
Proposed	✓	**heterogeneous**	✓	**Acc. config + DVFS**

D. Related Work

Optimized resource management of multi-DNN workload on resource-constrained devices requires awareness of resource heterogeneity, latency, system energy, and output accuracy. A few works have proposed accuracy-aware resource management of multi-application workloads on CPUs [11], and GPUs [12] without considering DNN workloads. Pipeit [3] proposed pipelining the DNN workload on CPU cores to enhance the inference throughput in batch applications. OmniBoost [9] scaled the DNN pipelining solution to both CPU, and GPU platforms. However, OmniBoost does not consider the impact of each decision on the energy or accuracy of the workload. MOC [9] has recently proposed joint optimization of energy and latency through DVFS and DNN pipelining of single DNN workloads. Band [2] used heuristics to Figure out the efficient workload placement on CPU, GPU, and Neural Processing Unit (NPU) platforms to optimize energy and latency of multi-DNN workloads. Similarly, Kim et. al. [10] presented energy-aware resource allocation of scenario-based multi-DNN workloads on all CPU, GPU, and NPU platforms. However, both [2], [10] are limited by the device resources due to a lack of accuracy configuration. Kang et. al. [6] performed

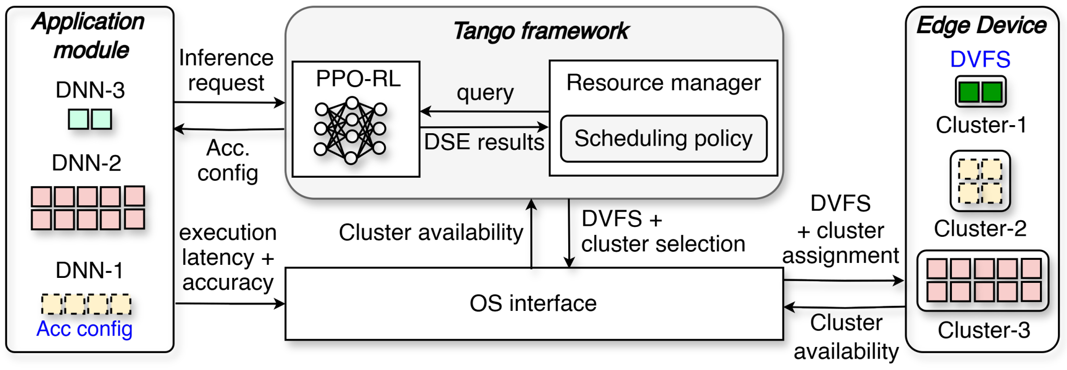

Fig. 5. Tango framework. (a). The resource manager receives the inference requests, reads the cluster availability, and invokes the RL for DSE, (c). ThePPO-based RL agent sends back the optimal cluster, and accuracy configuration, (d). the resource manager configures the accuracy of DNN-1 DNN, allocates cluster-2 to DNN-1, cluster-3 to DNN-2, and cluster-1 to DNN-3. The latency of DNN-3 is fine-tuned with DVFS.

layer-level quantization of a single DNN kernel with DNN pipelining and DVFS. The strategy performs static optimization where the GPU kernels require 16-bit quantizations leading to scalability limitations. Table-I compares our proposed solution *Tango* to the State-of-the-Art resource management solutions for DNN workloads on embedded platforms. We present a generalized and scalable resource allocation solution for multi-DNN workloads on embedded platforms to minimize inference latency while considering system energy minimization and output accuracy improvement.

III. Tango Framework

A. Framework Overview

We have designed *Tango*, a run-time resource management framework to infer dynamic multi-DNN workloads on Heterogeneous Multi-Processing (HMP)-based edge devices, hosting CPU and GPU clusters. Figure 5 shows an overview of the proposed *Tango* framework. *Tango* comprises of (i) *Application module* with the DNN application queue, (ii) *Resource Manager* hosting the scheduling policy, (iii) *PPO-RL* – a pre-trained PPO-based RL agent for run-time Design Space Exploration (DSE), and (iv) Linux-based Operating System (OS) interfaces to exchange data and decisions between different modules. The *Resource manager* receives inference requests from the *Application module* with multiple DNN applications ($App_0, App_1, App_2, ..., App_n \in Apps$); each DNN request has specific accuracy (A) and latency (L) requirements. The *Resource manager* gets the cluster availability and sends a scheduling query including the inference requests and the cluster availability to the RL agent. The RL agent is pre-trained with the average latency data of each DNN and its quantized variants across different clusters. The RL agent follows PPO protocol to find the optimal cluster and accuracy configuration for the given DNN applications. Finally, the *Resource Manager* enforces the accuracy configuration of the DNN applications and allocates the selected clusters. The *Resource Manager* monitors the applications' latency; if required, it fine-tunes the cluster performance by actuating DVFS. The *Tango* framework is agnostic of the hardware platforms and the architecture of the DNN workloads, providing a scalable and generalized solution for multi-DNN workload scheduling on edge devices.

Platform. A Linux-based OS runs on the device to enable an interface for workload scheduling, mapping, cluster selection, and run-time DVFS. The OS orchestrates application execution through resource allocation by binding the application process to the specified clusters. The OS uniformly distributes the parallelized tasks over multiple cores of a cluster to ensure a balanced execution. *Tango* uses the OS interface to override the Linux scheduler and maps a given application to the cores of a selected cluster. The OS interfaces provide granular control over cluster-level DVFS actuation through supported Linux drivers. The voltage and frequency values are organized as a tuple; the OS driver only allows frequency selection and automatically updates the corresponding voltage level. The device supports run-time power monitoring through onboard power sensors to analyze the impact of DVFS actuation on the energy budget of the device. For devices without power sensors, external power sensing equipment can be used to monitor run-time power consumption.

Workloads. *Tango* framework supports concurrently running DNN inference workloads of streaming applications. The applications perform vision tasks that generate batches of input images, with dynamic per-batch accuracy and latency requirements. The applications infer the DNN kernels with input data obtained from external devices such as cameras to perform cognitive tasks of classification, detection, tracking, etc. In this work, we used (i) VGG, (ii) MobileNetV2, (iii) ResNet, (iv) DenseNet, and (v) EfficientNet as the target DNNs. The Application Module loads the drivers and kernels required for the selected model's inference and stores the resulting latency and accuracy.

Accuracy configuration. *Tango* performs DNN accuracy configuration when the cluster selection and the DVFS actuation cannot jointly meet the latency requirements. The error-resilient nature of the DNN kernels can be exploited to reduce the latency by switching between quantized models of a given DNN kernel. Figure 6 shows how the *Resource*

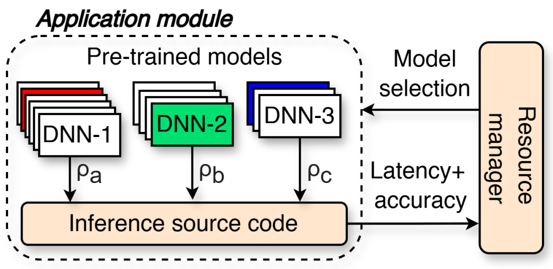

Fig. 6. Selecting quantized models for accuracy configuration of DNNs

manager chooses between a range of pre-trained models $(\rho_a, \rho_b, \rho_c, ..., \rho_m)$ at run-time for a given DNN workload. These pre-trained quantized variants of the DNN applications expose a diverse latency-accuracy trade-off space. The number of pre-trained models for each application can vary depending on the DNN architecture and the available storage space of the device. The selected models are loaded into the source code having supporting libraries and the environment designed for inference. The *Resource manager* explores the multiple model selection options such that the target latency is met on the selected cluster at tolerable accuracy loss.

Resource manager. The *resource manager* is implemented with a feedback loop as a control system. It integrates an internal policy continuously executed with a predefined frequency and performs three phases; (i) *Observe*: to monitor the status of both the applications and the architecture, (ii) *Decide*: to map the workload applications, and (iii) *Calibrate*: to actuate DVFS tuning. The complete workflow of the Resource manager's policy is explained in the next section.

B. Policy Workflow

The *Resource manager* policy is explained in Algorithm, 1. In the *Observe* phase, the policy updates the application queue on the arrival of a new application (Lines 2–4) and transitions to the *Decide* phase (Line 5). Conversely, if an application leaves the system after execution, the policy updates *Application queue* and transitions to the *Calibrate* phase (Lines 6–8).

In the *Decide* phase, the policy iterates through all the applications in the *Application queue* and stores their accuracy and latency requirements (Lines 9–10). The policy explores the optimal mapping and accuracy configuration of all the applications running, as the arrival of a new application changes the total workload. The policy reads the cluster availability to observe the current availability of CPU and GPU clusters (Line 11). A cluster is considered unavailable for allocation if it is already busy executing a workload. The policy invokes DSE through the PPO-based RL agent to figure out the optimal resource allocation and accuracy configuration of the workload following the cluster availability and application requirements (Line 12–14). The design of the PPO-based RL agent is explained in section III-C. The selected DNN models are mapped to the selected clusters (Line 15) and the policy transitions to the *Calibrate* phase for DVFS tuning.

In *Calibrate* phase, the policy iterates through all the running applications and adjusts the cluster DVFS according

Algorithm 1 Policy workflow of Resource manager

```
1: Input: Apps
   Phase: Observe
2: if NewApp then
3:     Update the Application Queue: Apps
4:     NewApp ← 0
5:     Phase ← Decide
6: else if App.left then
7:     –Apps.queue
8:     Phase ← Calibrate
   Phase: Decide
9: for all $app_i \in$ Apps do
10:    App_reqs_list[] ← ($app_i.acc_{req}$, $app_i.latency_{req}$)
11: cluster_status ← get_cluster_availability()
12: config ← run_DSE(cluster_status, Apps, App_reqs_list)
13: new_maps[] ← config.mappings[]
14: selected_models[] ← config.models[]
15: enfore_new_mapping(selected_models[], new_maps[])
16: Phase ← Calibrate
   Phase: Calibrate
17: for all $apps_i \in$ Apps do
18:    $f_{est} \leftarrow \frac{app_i.latency_{req}}{app_i.latency_{curr}} * f_{curr}$
19:    update_DVFS($f_{est}$, $app_i$.cluster)
20: Phase ← Observe
```

to the target latency. The policy reads the current latency of a DNN application and calculates a latency factor as the ratio of the current to the required latency. The new DVFS frequency level is calculated by multiplying the latency factor with the current cluster frequency of the given application (Line 18). Finally, the policy enforces the new DVFS settings to minimize the latency according to the application requirements (Line 19) and transits to the *Observe* phase (Line 20).

C. Design Space Exploration (DSE) Strategy

We design a DSE strategy for minimizing the latency of the DNN inference requests, by considering the heterogeneity of edge devices, latency, and accuracy requirements of workloads. We address the edge inference problem using RL that is widely adopted for intelligent decision-making in complex systems, using information collected over time. We use Markov's Decision Process (MDP) based RL exploration to resolve the multi-DNN inference problem. The MDP paradigm typically contains state, action, and reward spaces, representing the current environment status, a selected action, and the efficacy of a selected action in a given state. Through extensive training, RL explores different state-action combinations and converges towards an optimal policy. While conventional RL typically optimizes one objective, the multi-DNN inference problem has multiple conflicting objectives of latency minimization at minimal accuracy loss.

In *Tango* framework, we map (i) edge cluster availability, accuracy, and performance requirements of workloads to the *State space*, (ii) cluster selection and accuracy configuration to the *Action space* and (iii) minimizing latency and maximizing accuracy to the *Reward space*. Figure 7 shows the RL agent is designed for making scheduling decisions in the *Tango* framework. The RL agent is modeled as a tuple (S, A, P, ω),

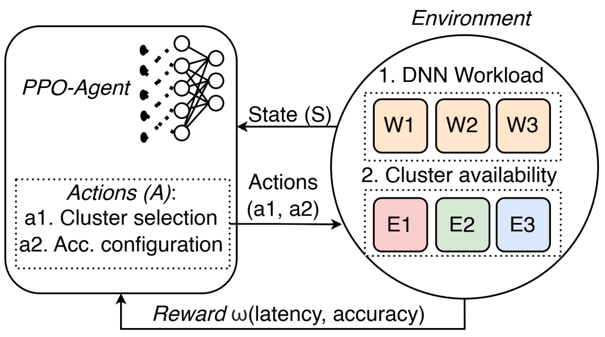

Fig. 7. Proximal Policy-based reinforcement learning agent model.

where S represents the state space, A is the action space, P is the state transition probability matrix, and R is the immediate reward function of G objective functions. The environment is denoted by $S(E, W)$, where S shows end-to-end system, E shows edge node resources, and W shows DNN workloads. The availability of the heterogeneous clusters is represented as $E = \{E_1, E_2, .., E_n\}$, where E_i is a binary integer representing the busy status of the available CPU or GPU clusters. DNN workloads are represented as $W = \{W_1, W_2, .., W_n\}$, where W_i is an inference workload request consisting of a tuple $(perf, acc)$ that represents performance and accuracy requirements. These action space A includes cluster selection:

$$a1 : map(Ei \leftarrow W) \tag{1}$$

and accuracy configuration:

$$a2 : select(W \leftarrow \rho) \tag{2}$$

For accuracy configuration (ρ), the optimal model variant is selected to perform inference from a pool of pre-trained models, including quantized, pruned, and heavy models. Each model presents different performance and accuracy characteristics depending on their depth and number of parameters. The reward (ω) depends on the latency and accuracy constraints:

$$\omega = \alpha \cdot A - \beta \cdot L \tag{3}$$

where α and β are the accuracy (A) and latency (L) coefficients, respectively, their values depend on the application scenario. In MDP, the agent follows a policy function π, where π is a $S \times A \rightarrow$ [0, 1] mapping, i.e., for a given state, the selected action corresponds to a certain probability distribution. In our baseline multi-DNN inference problem, the state and action sets are discrete but too large for manual selection. Thus, the agent uses reinforcement learning to study the value of each policy. In this task, we use the PPO algorithm to provide the probability distribution of actions given observations in the environment. The loss function in PPO, known as the clipped surrogate objective, evaluates policy changes by comparing new and old policies while incorporating action advantages. Gradient descent minimizes this loss to update policy network parameters, maximizing the objective function. States, actions, and rewards are stored in a buffer, facilitating policy network updates based on past experiences, essential for optimizing cluster mapping for future inference requests.

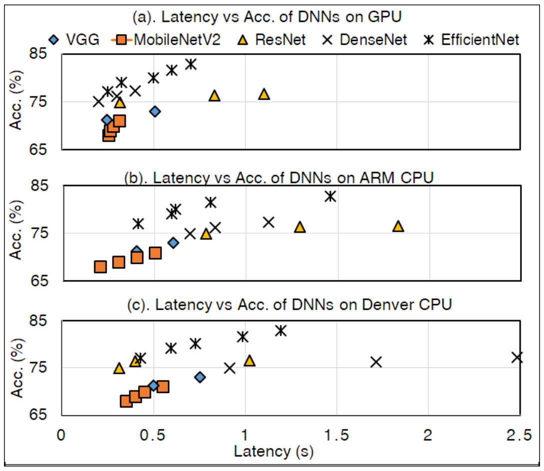

Fig. 8. The accuracy-latency trade-off space of different DNN models

$$L(\theta) = E[min(r(\theta) * v, clip(r(\theta), 1 - \epsilon, 1 + \epsilon) * v] \tag{4}$$

where r(θ) is the probability ratio between two subsequent policies, v is the advantage function measuring the advantage of an action compared to the average action taken in that state, and ϵ is a hyper-parameter controlling the clipping range. We executed different DNN kernels in each of their quantized variants on the target platform [7] to collect latency, power consumption, and accuracy. We used this data to train the RL agent. The pre-trained RL is deployed in the *Tango* framework.

IV. Experimental Evaluation

A. Experimental Setup

Platform and middleware. For evaluation, we use Jetson TX [7] heterogeneous edge platform including Pascal GPU, quad-core ARM CPUs, and dual-core Denver CPUs. The device has onboard power sensors for collecting run-time power consumption of CPU and GPU clusters. The device hosts Linux-18.04 OS with Cuda support to enable GPU operations. We have implemented the *Tango* framework (shown in Figure 5) as a middleware in Python, with a source code of 200 lines. The middleware enforces cluster selection, accuracy configuration, and DVFS. The decisions of *Tango* framework are made by the RL agent. We implemented the RL agent using the Gymnasium library [13]. The *Tango* framework uses CGroup libraries to map a DNN application to the required cluster and apply cluster-level DVFS settings at run-time. The maximum operating frequency of the Pascal GPU is 1.12GHz, the ARM cluster is 2GHz, and the Denver cluster is 2GHz, with frequency scaling in steps of 100Hz.

Workloads and accuracy configuration. For experiments, we considered widely used image classification DNNs [14] including, (i) VGG, (ii) MobileNetV2, (iii) ResNet, (iv) DenseNet, and (v) EfficientNet. These models are highly tunable with a wide energy-latency space, catering to diverse real-world scenarios. All of these models are pre-trained on the ImageNet dataset [15]. We use the TensorFlow-GPU library

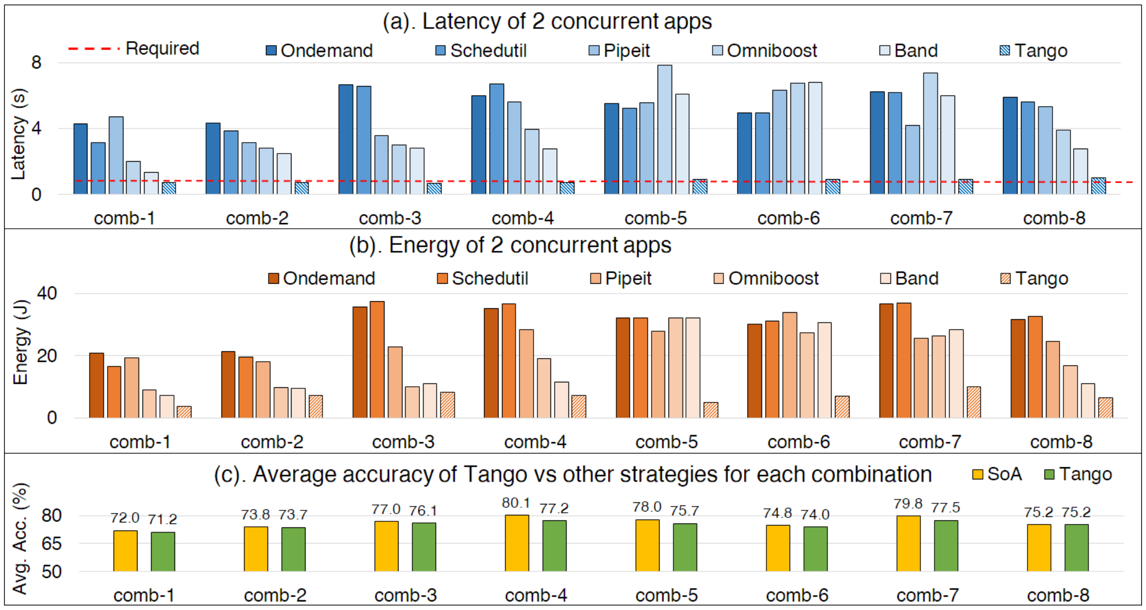

Fig. 9. Comparison of (a). latency, (b). energy (c). accuracy of Tango against other strategies while running different combinations of 2 concurrent DNNs.

to infer the pre-trained models from the Keras application library [14]. We trained the RL agent offline with the latency and accuracy of these DNN applications and their quantized variants. We meet the required accuracy by using a pre-trained model selection from the pool of DNN kernels. Further, Figure 8 shows the accuracy and latency trends of the base DNNs and their lighter variants on all clusters of Jetson TX. The accuracy-latency trend shows that typically the models are biased towards GPU. Secondly, the ARM cluster reports lesser latency than Denver for most DNNs. For evaluation metrics, we use (i) *latency* as the time taken per inference, (ii) top-1% output *accuracy* by comparing predicted labels to genuine labels and calculating the proportion of correctly classified samples, and (iii) *energy* as the product of latency and the average power consumption per inference.

Comparison w.r.t. state-of-the-art approaches. We evaluated our proposed strategy against default Linux governors including `Ondemand` and `SchedUtil`. Moreover, we compared against state-of-the-art multi-DNN scheduling approaches including Pipeit [3], OmniBoost [4], and Band [2]. Pipeit essentially pipelines the DNN inference over CPU clusters to gain higher throughput. To implement Pipeit, we converted the DNN kernels to Directed Acyclic Graphs (DAG) and performed vertical partitioning of the graph nodes. We implemented the performance prediction model of Pipeit and scheduled the DNNs over ARM and Denver cluster. OmniBoost [4] presents a DNN pipelining strategy over CPU and GPU clusters using a Monte-Carlo tree-based throughput estimator. Again we implemented the strategy by vertically partitioning the DAG-based DNN models and mapped them over the CPU and GPU clusters. Finally, Band [2] minimized the latency by jointly considering DNN pipelining and DVFS for CPUs, and GPUs. We enhanced these strategies to adapt to the varying workload conditions as in real-world application scenarios.

B. Experimental Results

Experiments with multi-DNN workloads We compared the performance of *Tango* against different strategies in multi-DNN scenarios with two and three concurrently running applications. For each experiment, we initiate simultaneous DNN requests to create concurrent workload scenarios. For evaluation, we used a maximum of three concurrent DNNs for the thermal safety of the evaluation platform while covering the majority of real-world scenarios. However, the *Tango* framework can handle a generic number of concurrent DNNs, as practically supported by the edge platform.

Latency comparison for 2 concurrent applications. We evaluated the latency and energy of each strategy by concurrently running two DNN applications in eight different combinations as shown in Figure 9. We choose two different DNNs and generate simultaneous inference requests for both kernels at run-time. Figure 9a shows the target latency of the workload and the latency achieved by each strategy across all combinations. For this experiment, we set the target latency to 1s following the practical use of vision DNNs in edge and mobile platforms. Similarly, Figure 9b shows the energy consumption of all strategies across all workload combinations. The Ondemand and SchedUtil governors randomly place one application on GPU and the other on the ARM cluster as these clusters have relatively high-performance capabilities. The greedy resource allocation approach of the governors leads to higher energy consumption without meeting the required latency of the workloads. Pipeit [3] schedules a pipelined inference of the two applications on the ARM and Denver clusters without meeting the target latency. Omniboost [4] opportunistically schedules a pipelined workload over all clusters depending on the throughput estimations. Still, the workload placement and pipelining are inefficient in meeting the intense latency requirements for all the combinations. Band [2] takes leverage

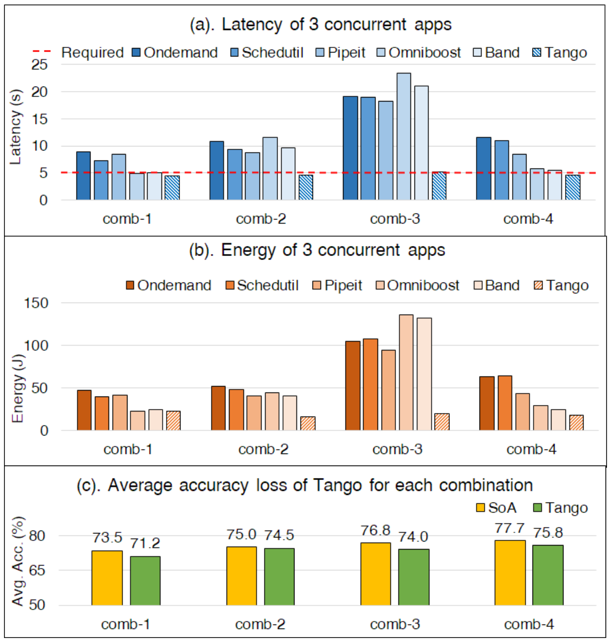

Fig. 10. Comparison of (a). latency, (b). energy (c). accuracy of Tango against other strategies while running different combinations of 3 concurrent DNNs.

of DVFS to minimize the latency further and reports the least latency against the other State-of-the-Art (SoA) approaches. However, the strategy reports higher energy consumption in multiple combinations due to high power consumption and is still unable to achieve the target latency. Finally, *Tango* (i) ensures coordinated inference by exploring the gain of placing each application to different clusters, (ii) leverages latency minimization through accuracy configuration at minimum accuracy loss, and (iii) fine tunes the final selection with DVFS. Figure 9c shows the average accuracy of each combination for all strategies. For all eight combinations, *Tango* improves the average latency against the strategies reporting the least latency by 44.10%, 71.54%, 75%, 73.02%, 85.18%, 86.56%, 84.59%, and 63.96 respectively. Similarly, *Tango* consumes less energy consumption on average than other strategies 47.89%, 23.62%, 25.49%, 36.77%, 84.31%, 77.55%, 64.54%, and 40.75% respectively. The average accuracy loss by *Tango* over the eight combinations is 1.3%.

Latency comparison for 3 concurrent applications. We evaluated the latency and energy of each strategy for different combinations of three concurrently running applications as shown in Figure 10. Running three applications concurrently leads to a significant increase in the inference latency on each cluster due to inter-application interference, memory operations, and resource contention. Thus, we set the target latency to 5s. With 3 concurrent DNNs each strategy is forced to allocate workload to the low-performing Denver cluster. Hence, these strategies suffer significant performance loss during the workload execution on the Denver cluster. For example, the inference latency of all strategies except *Tango* in combination-3 is more than 15 seconds, violating the target by x3. Here, *Tango* opportunistically configures the accuracy of the DNN kernels and achieves the target latency successfully. *Tango* improves the latency against the compared strategies by 9.82%, 52.17%, 75.48%, and 17.66% for each combination respectively. Similarly, *Tango* improves energy consumption by 8.44%, 59.46%, 85.21%, and 27.08% respectively. The average accuracy loss over the four combinations of three concurrently running applications is 1.69%. Overall, *Tango* improves the latency and energy consumption by 61%, and 48.4% for all the experiments at an accuracy loss of 1.9%.

V. Conclusions

We presented *Tango* framework for low latency multi-DNN inference on heterogeneous edge platforms. Our approach jointly configures cluster selection, accuracy configuration, and DVFS for minimizing inference latency. We evaluated *Tango* on the Jetson platform, achieving latency and energy improvements of 61%, and 48.4% at a minimal accuracy loss of 1.59% against state-of-the-art. We consider DNN partitioning and distribution for future work.

Acknowledgements

This work is funded by the European Union's Horizon 2020 Research and Innovation Program (APROPOS) under the Marie Curie grant No. 956090

References

[1] C.-J. Wu *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 331–344.
[2] J. Seong *et al.*, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022.
[3] S. Wang *et al.*, "High-throughput cnn inference on embedded arm big. little multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, 2019.
[4] A. Karatzas *et al.*, "Omniboost: Boosting throughput of heterogeneous embedded devices under multi-dnn workload," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. USA: ACM/IEEE, 2023.
[5] J. Kim *et al.*, "Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints," *IEEE Transactions on Computers*, 2022.
[6] D. Kang *et al.*, "Joint optimization of speed, accuracy, and energy for embedded image recognition systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018.
[7] NVIDIA, "Jetson tx2 module," 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2
[8] Z. Zhuoran *et al.*, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[9] Y. Wu *et al.*, "Moc: Multi-objective mobile cpu-gpu co-optimization for power-efficient dnn inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–10.
[10] S. Minakova *et al.*, "Scenario based run-time switching for adaptive cnn-based applications at the edge," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 21, no. 2, pp. 1–33, 2022.
[11] A. Kanduri *et al.*, "Approximation-aware coordinated power/performance management for heterogeneous multi-cores," in *ACM Design Automation Conference (DAC)*, 2018, pp. 1–6.
[12] A. Miele *et al.*, "A runtime resource management and provisioning middleware for fog computing infrastructures," *ACM Trans. Internet Things*, vol. 3, no. 3, apr 2022.
[13] M. Towers *et al.*, "Gymnasium," 2023. [Online]. Available: https://zenodo.org/record/8127025
[14] Keras Team. (Accessed: 2024) Keras documentation. [Online]. Available: https://keras.io/api/applications/
[15] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.