TURUN
YLIOPISTO
UNIVERSITY
OF TURKU

Efficient Run-time Systems for Edge AI Inference

Zain Taufique

TURUN YLIOPISTON JULKAISUJA – ANNALES UNIVERSITATIS TURKUENSIS
SARJA – SER. F OSA – TOM. 88 | TECHNICA – INFORMATICA | TURKU 2026

TURUN
YLIOPISTO
UNIVERSITY
OF TURKU

EFFICIENT RUN-TIME SYSTEMS FOR EDGE AI INFERENCE

Zain Taufique

TURUN YLIOPISTON JULKAISUJA – ANNALES UNIVERSITATIS TURKUENSIS
SARJA – SER. F OSA – TOM. 88 | TECHNICA - INFORMATICA| TURKU 2026

University of Turku

Faculty of Technology
Department of Computing
Information and Communication Technology (ICT)
The Doctoral Programme in Technology (DPT)

Supervised by

Dr. Anil Kanduri
University of Turku
Finland

Prof. Pasi Liljeberg
University of Turku
Finland

Prof. Amir M. Rahmani
University of California, Irvine
United States of America

Reviewed by

Prof. Gianluca Palermo
Politecnico di Milano
Italy

Prof. Muhammad Shafique
New York University Abu Dhabi
United Arab Emirates

Opponent

Prof. Mario Di Francesco
Aalto University
Finland

The originality of this publication has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

ISBN 978-952-02-0753-3 (PRINT)
ISBN 978-952-02-0754-0 (PDF)
ISSN 2736-9390 (PRINT)
ISSN 2736-9684 (ONLINE)
Painosalama, Turku, Finland, 2026

Dedicated to my late grandfather, Mr. Rafique Sufi,
who always believed that I would achieve excellence in life

UNIVERSITY OF TURKU
Faculty of Technology
Department of Computing
Information and Communication Technology (ICT)
TAUFIQUE, ZAIN: Efficient Run-time Systems for Edge AI Inference
Doctoral dissertation, 158 pp.
The Doctoral Programme in Technology (DPT)
June 2026

ABSTRACT

Efficient orchestration of AI inference on heterogeneous edge platforms is crucial for meeting the stringent latency, energy, and accuracy requirements of edge AI applications. The edge devices integrate heterogeneous compute clusters, including CPUs, GPUs, and Neural Processing Units, each exhibiting asymmetric energy-performance characteristics. Modern AI workloads handle diverse runtime inference requests that arrive both continuously and in response to user prompts, each with distinct latency, accuracy, and priority requirements. Despite the availability of heterogeneous computational resources, existing scheduling mechanisms remain primarily conservative, failing to utilize them and violating workload and system constraints. This dissertation addresses the scheduling challenges of inferring multiple AI workloads on resource-constrained edge platforms.

This dissertation develops an approximation-aware runtime resource management approach that jointly optimizes power, performance, and accuracy under strict energy constraints on heterogeneous edge platforms. The proposed *TangoX* framework targets systems with asymmetric CPU and GPU clusters, using a reinforcement-learning-based orchestrator to coordinate cluster assignment, DVFS, and model precision to reduce latency while constraining accuracy degradation. To support compound AI workloads, *Twill* extends runtime coordination across heterogeneous CPU, GPU, and DLA clusters, integrating vision, transformer, and language models while enforcing task prioritization and cluster affinity to satisfy latency requirements within power budgets. A distributed edge framework further enables adaptive workload distribution by partitioning data across devices, balancing latency and accuracy under dynamic conditions. Complementing this, *HiDP* introduces hierarchical DNN partitioning across global and local edge layers, enabling scalable and coordinated inference. All frameworks are implemented on heterogeneous edge hardware and evaluated using diverse workloads. Results demonstrate consistent reductions in inference latency and energy consumption compared with state-of-the-art techniques, with minimal impact on model accuracy. Collectively, these contributions advance the design of efficient, adaptive, and scalable edge AI systems.

KEYWORDS: DNN, Inference, LLM, Edge AI, Transformers, Energy, Performance

TURUN YLIOPISTO
Tiedekunta
Laitos
Oppiaine
TAUFIQUE, ZAIN: Efficient Run-time Systems for Edge AI Inference
Väitöskirja, 158 s.
Tohtoriohjelma
Kuukausi 2026

TIIVISTELMÄ

AI-inferenssiprosessien tehokas orkestrointi heterogeenisissä edge platform -ympäristöissä on keskeistä, jotta edge AI -sovellusten tiukat latenssi-, energia- ja tarkkuusvaatimukset voidaan täyttää. Edge-laitteet sisältävät heterogeenisiä compute cluster -rakenteita, mukaan lukien CPU:t, GPU:t ja Neural Processing Unit (NPU) -yksiköt, joilla on epäsymmetriset energia-suorituskykyominaisuudet. Modernit AI-workloads tuottavat monimuotoisia runtime inference request -pyyntöjä, jotka saapuvat jatkuvasti ja käyttäjän prompt-syötteiden seurauksena erilaisilla latenssi-, tarkkuus- ja prioriteettivaatimuksilla. Huolimatta heterogeenisten computational resources -resurssien saatavuudesta nykyiset scheduling mechanisms -mekanismit ovat pääosin konservatiivisia eivätkä kykene hyödyntämään niitä tehokkaasti, mikä johtaa workload- ja system constraint -rajoitteiden rikkomiseen. Tämä dissertation käsittelee useiden AI workload -tehtävien scheduling-haasteita resource-constrained edge platform -ympäristöissä.

Tämä dissertation kehittää approximation-aware runtime resource management -lähestymistavan, joka optimoi samanaikaisesti tehonkulutus-, suorituskyky- ja tarkkuustekijöitä heterogeenisissä edge platform -järjestelmissä. Ehdotettu *TangoX*-framework hyödyntää reinforcement learning -pohjaista orkestroijaa cluster assignment-, DVFS- ja model precision -asetusten koordinoimiseksi latenssin pienentämiseksi samalla kun tarkkuuden heikkeneminen pidetään hallinnassa. Compound AI workload -kokonaisuuksien tukemiseksi *Twill* laajentaa runtime coordination -mekanismia heterogeenisiin CPU-, GPU- ja DLA cluster -ympäristöihin integroiden vision-, transformer- ja language model -mallit sekä enforcing taskien priorisointi- ja cluster affinity -periaatteet latenssivaatimusten täyttämiseksi power budget -rajojen sisällä. Distributed edge framework mahdollistaa adaptiivisen workload distribution -menetelmän partitioning data across devices -periaatteen avulla tasapainottaen latenssi- ja tarkkuustekijöitä dynaamisissa olosuhteissa. Tätä täydentää *HiDP*, joka esittelee hierarchical DNN partitioning -ratkaisun global- ja local-edge layers -tasojen välillä mahdollistaen skaalautuvan inferenssisuorituksen. Kaikki frameworkit on toteutettu heterogeenisille edge hardware -alustoille ja arvioitu monipuolisilla workload-kokonaisuuksilla. Tulokset osoittavat johdonmukaisia parannuksia inferenssilatenssi- ja energiankulutusmittareissa verrattuna state-of-the-art techniques -menetelmiin samalla kun vaikutus mallien tarkkuuteen pysyy minimaalisena.

ASIASANAT: DNN, päättely, LLM, reunalaskenta, transformerit, energia

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my primary supervisor, Dr. Anil Kanduri. From the very first day, he believed in me, challenged my thinking, and encouraged me to see my true potential. Anil continuously motivated me to grow both professionally and personally, always pushing me to aim higher and think creatively. He introduced me to the world of independent research and taught me how to take ownership of my ideas. This thesis is deeply shaped by his guidance, mentorship, and unwavering support throughout my journey. I am also thankful to Dr. Pasi Liljeberg for his unwavering support throughout my doctoral studies, including research, coursework, travel, and personal matters. I am grateful to Dr. Amir M. Rahmani for being a part of my research journey. I would also like to acknowledge the efforts of Dr. Antonio Miele, who consistently prioritized my research, often placing it ahead of his own commitments. I am deeply grateful to my close team member and research partner, Aman Vyas, whose collaboration truly embodied the spirit of teamwork and made this journey immensely rewarding. Aman has been a constant source of support through every high and low, always standing by me with encouragement and sincerity. Working alongside him has been one of the most meaningful parts of this journey. I sincerely hope he achieves even greater success in his PhD journey than I ever could. I would like to thank my associate, Dr. Arman Azanpour, for guiding me in understanding the geometry of schematic diagrams, which I consistently applied throughout my research. I want to give a shoutout to my team members in the Health Technology Lab, with whom I shared enjoyable memories, including Ajdar Ullah, Ismail Elnaggar, Kianoush Kazemi, Katri Karhinoja, Yangyang Zhao, Sepehr Seifizarei, Fatemeh Sarhaddi, Nosiba Khougali, Tomi Jaakola and Jiawei Yang. I would also like to thank my friends Shaheryar Tariq and Shaheryar Malik at Abo Akademi for helping me set up the Linux dependencies on the hardware boards I relied on from the very beginning of my doctoral studies.

This research was supported by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS). I am thankful to the APROPOS project leaders, Dr. Aleksandr Ometov and Dr. Jari Nurmi from Tampere University, for their leadership and support. Special thanks to Dr. Dewant Kantare for his help with the ideation of DNN partitioning, which made a significant contribution to my research. I also acknowledge Bosch Sensortec for hosting my industrial secondment as part of APROPOS.

I would like to thank the PTC group, where friends became my family. Starting with future doctors, Waseem Haider and Naman Naushahi, who were the first people in Turku to take me home when I was freezing at -27 degrees at the bus station. I also met Intekhab, who was just as scared of the Finnish weather as I was. But once he adapted to the Finnish lifestyle, he became one of the most dependable friends in my circle. Then I met Asim Nadeem and Shaheryar Malik, who became my closest friends, spending days and nights with me, talking me through my tough times and cheering me up with long debates and discussions about the most useless yet creative topics. Then came Adil and Basit, who pushed me out of my home and showed me what life is actually about, living in Finland. I believe that the long discussions with Asim, Shaheryar and Adil actually helped me with my research skills and creative thinking. I also met Dr. Hassan Qureshi, who taught me that one can truly enjoy both research and family life. He showed me an example of fatherhood, and I have always been impressed by his deep knowledge of research while still always being there for his wife and two sons. Later came Ikram and Shaheryar Tariq with their board games, and a lot of fun discussions. Ikram was there for me at the lowest point of my life. He taught me that simply showing up when someone needs support can mean the world to them. I met Shafi Ud Din, who taught me restraint and humility while always being supportive. I met Emad and Ismail, who added a lot of fun and joy to my life with their songs and startup talks. I met Hassan Qadeer and Mihail Ruotsi, who challenged my political and religious biases while helping me enjoy life through new experiences. Once again, I want to mention Naman Shah G again for providing us with a space to relax and part, which helped me keep my sanity.

I am deeply grateful to my father, Taufique Ur Rehman, my mother, Shazia Taufique, and my sister, Zarin, for their unwavering love and support throughout this journey. I would also like to sincerely thank my in-laws, Dr. Naveed Akhter, Mrs. Farhana, and Munem, for their encouragement and kindness. The unwavering support of both my family and my in-laws, whether through their presence or even from afar, gave me the strength to hold myself together. Words feel insufficient to express how much their love, faith, and existence kept me going through this four-year journey of living and working away from home. Without them, this journey would not have been possible. Finally, my heartfelt thanks go to my wife Rameesha, whose belief in me never wavered, who supported me throughout this journey, and who shared the greatest excitement when I was selected for the doctoral program. Whatever I say about Rameesha's contribution to my PhD will never be enough. My whole life truly revolves around her, and there is no better feeling than sharing my time and journey with her. I am deeply grateful to my daughter, Rekhta, for simply existing, for her smiles, and for the joy and motivation she brings to my life every day. I hope that when you see my thesis, you feel that your baba made you proud.

11-05-2026
Zain Taufique

ZAIN TAUFIQUE

Zain Taufique holds a Master's Degree in Electronics and Embedded Systems from Lahore University of Management Sciences (LUMS), Lahore. His research focuses on adaptive edge AI inference on resource-constrained, heterogeneous platforms. He has collaborated with Bosch Sensortec during an industrial secondment, contributing to research on low-power scheduler design. As an early-stage researcher, he was awarded a research grant under the Horizon 2020 ITN Marie Skłodowska-Curie program to participate in the Approximate Computing for Power and Energy Optimization (APROPOS) project.

Table of Contents

Abbreviations

API	Application Programming Interface
AI	Artificial Intelligence
AR	Augmented Reality
cAI	Compound Artificial Intelligence
CNN	Convolution Neural Network
CPU	Central Processing Unit
CS	Context Switching
DAG	Directed Acyclic Graph
DL	Deep Learning
DLA	Deep Learning Accelerator
DNN	Deep Neural Network
DoP	Degree of Parallelism
DSE	Design Space Exploration
DVFS	Dynamic Voltage/Frequency Scaling
FCFS	First-come-first-serve
FE	Feature Extraction
FPS	Frames Per Second
FSM	Finite State Machine
GN	Global Node
GPU	Graphics Processing Unit
HMP	Heterogeneous Multi-Processors
HSA	Heterogeneous System Architecture
ILP	Integer Linear Programming
IoT	Internet of Things
KNN	K-Nearest Neighbors
kM	k-Means
LAN	Local Area Network
LLM	Large Language Model
LM	Language Model
LN	Local Node
LR	Linear Regression
LSTM	Long Short-Term Memory
MAC	Multiply–Accumulate operation

MAS	Multi-Agent Systems
MIG	Multi-Instance GPU
ML	Machine Learning
MPS	Multi-Process Service
MR	Mixed Reality
NPU	Neural Processing Unit
NVDLA	NVIDIA Deep Learning Accelerator
ONNX	Open Neural Network Exchange
OS	Operating System
PPO	Proximal Policy Optimization
PPW	Performance per Watt
QoS	Quality of Service
RL	Reinforcement Learning
RTM	Run-time Resource Management
SM	Streaming Multiprocessor
SoA	State-of-the-Art
SoCs	System-on-Chips
TDP	Thermal Design Power
THD	Total Harmonic Distortion
TM	Task Migration
ViT	Vision Transformer
VLM	Vision-Language Model
VR	Virtual Reality
WLAN	Wireless Local Area Network

List of Original Publications

This dissertation is based on the following original publications, which are referred to in the text by their Roman numerals:

I **Zain Taufique**, Anil Kanduri, Antonio Miele, Amir Rahmani, Cristiana Bolchini, Nikil Dutt, Pasi Liljeberg. *Exploiting Approximation for Runtime Resource Management of Embedded HMPs*. ACM Transactions on Embedded Computing Systems (ACM-TECS), 2024.

II **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Tango: Low latency Multi-DNN inference on heterogeneous edge platforms*. IEEE 42nd International Conference on Computer Design (ICCD), 2024.

III **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *TangoX: Robust Multi-DNN Inference on Heterogeneous Edge Platforms*. IEEE Transactions on Computers (IEEE-TC (under-revision)), 2026.

IV **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Twill: Scheduling Compound AI Systems on Heterogeneous Mobile Edge Platforms*. International Conference on Computer-Aided Design (ICCAD), 2025.

V **Zain Taufique**, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Adaptive workload distribution for accuracy-aware DNN inference on collaborative edge platforms*. 29th Asia and South Pacific Design Automation Conference (ASP-DAC), 2024.

VI **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *HiDP: Hierarchical DNN Partitioning for Distributed Inference on Heterogeneous Edge Platforms*. IEEE Design, Automation & Test in Europe Conference (DATE), 2025.

The original publications have been reproduced with the permission of the copyright holders.

The following peer-reviewed publications were published during doctoral studies. These publications are not included in this thesis but are closely related.

VII Hans Jakob Damsgaard, Antoine Grenier, Dewant Katare, **Zain Taufique**, et. al. *Adaptive approximate computing in edge AI and IoT applications: A review*. Journal of Systems Architecture (JSA), 2024.

VIII **Zain Taufique**, Anil Kanduri, Muhammad Awais Bin Altaf, Pasi Liljeberg, *Approximate feature extraction for low power epileptic seizure prediction in wearable devices*. IEEE Nordic Circuits and Systems Conference (NorCAS), 2021

1 Introduction

Artificial Intelligence (AI) has become essential across diverse use cases in autonomous systems, healthcare, mobile computing, and mixed reality for enabling perception [1], independent decision-making [2], and context-driven task execution [3]. Modern AI applications are increasingly deployed on edge devices, such as smartphones [4], Augmented Reality (AR)/Virtual Reality (VR) headsets [5], and smart glasses [6]. Edge AI enables robust on-device inference closer to the source, providing fast responses while preserving user data privacy. The edge platforms integrate heterogeneous compute clusters including CPUs, GPUs, and Neural Processing Units (NPUs), each exhibiting asymmetric energy and latency characteristics. Advanced edge AI applications require inference of diverse machine learning models, including Deep Neural Networks (DNNs), encoder transformers, and Large Language Models (LLMs) to perform multiple cognitive tasks, such as classification, detection, understanding, and reasoning [7]. Efficient run-time inference frameworks are a foundational requirement for supporting dynamic AI workloads in resource-constrained edge environments.

1.1 Challenges of Edge AI inference

AI workloads can generate simultaneous inference requests for multiple models with varying latency and accuracy requirements, driven by the application workflow and user preferences. However, edge platforms are fundamentally resource-constrained, with limited computational capacity and tight energy budgets [8; 9]. Moreover, Machine Learning (ML) workloads expose a controllable accuracy–latency trade-off space, enabling dynamic selection of different pruned and quantized model configurations to satisfy performance requirements. The arrival of inference requests is stochastic, leading to variations in run-time workload. The overarching challenge in edge-based inference is to develop efficient run-time frameworks that dynamically schedule inference workloads across diverse platforms to optimize energy, latency, and accuracy under varying system dynamics. Edge inference frameworks aim to minimize latency and energy consumption while preserving accuracy, enabling faster execution and a smoother user experience [10; 11].

Edge orchestration faces several resource-management challenges when dynamically deciding where and how to execute inference. Without adaptive coordination,

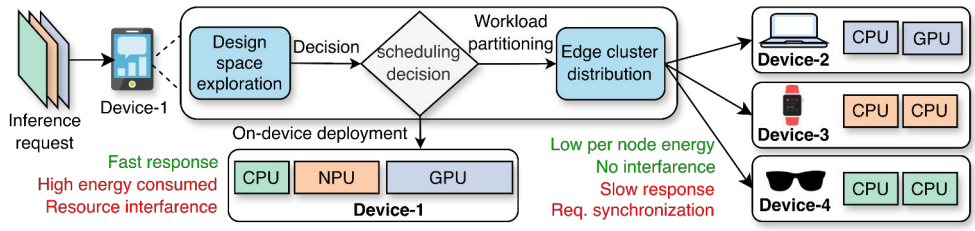

Figure 1. Performance and energy trade-off space between on-device and distributed inference.

inference workloads may underutilize available compute clusters and fail to meet latency, energy, and accuracy requirements. The challenges of orchestrating inference of multiple AI workloads on resource-constrained heterogeneous edge devices include:

- **Platform diversity:** The edge platforms have diverse computational clusters with asymmetric performance, energy consumption and operator-level support for executing different workload operations.
- **Workload Diversity:** The AI workloads have non-uniform latency, accuracy, and priority requirements based on the model architecture of vision and language models, including DNNs, transformers and LLMs.
- **Stochastic run-time variance:** The application's arrival is stochastic and unknown to the schedulers, depending on user prompts and workflow. Resource availability also varies at run-time due to multiple inference requests executing simultaneously.
- **Exploring the design space** Exploring the design space to find a suitable resource scheduling solution becomes complex as the number of applications and computational resources increases.

The edge AI inference challenges are illustrated in Figure 1. Each incoming inference request arrives with varying latency and accuracy requirements. The workload can either be executed on the heterogeneous compute clusters of the source device or partitioned and offloaded to neighboring trusted devices by forming an interconnected edge cluster. Both on-device deployment and distributed edge inference have their pros and cons. On-device inference has low communication overhead and execution latency but limited computational resources. Distributed inference provides access to a larger set of computational clusters, enabling high throughput, but it requires continuous information exchange and synchronization. The scheduling strategy must therefore decide how to execute the workload, either locally on the source device or by distributing it across the edge cluster, to satisfy application-level requirements under non-deterministic system dynamics. Hence, run-time de-

sign space exploration is required to support strategic decisions for efficient workload execution.

Existing edge AI frameworks are primarily conservative, failing to (i) efficiently utilize available resources, (ii) adapt to the changing system dynamics, (iii) scale to workload and platform heterogeneity, and (iv) service the application priorities based on user-defined urgencies [12; 13; 14; 9; 8; 15; 16; 17; 10]. Despite the availability of mature inference frameworks such as TensorRT [18], TensorFlow [19], PyTorch [20], and NCNN [21], these solutions primarily focus on model-level graph optimizations and backend-specific acceleration. They lack system-level awareness and run-time coordination mechanisms needed to manage multiple concurrent models under dynamic resource constraints and application-specific priorities. Modern edge inference strategies distribute the workload to nearby trustworthy devices via workload partitioning to increase the execution throughput [22; 23; 24]. However, workload distribution incurs significant communication overhead and requires careful synchronization of data and model weights across devices to obtain the final results. This dissertation presents robust, adaptive and scalable AI orchestration frameworks for jointly orchestrating multiple inference requests of diverse models with non-uniform requirements. An efficient solution for both on-device and distributed inference is presented, based on heuristic and Reinforcement Learning (RL) driven policies to determine a suitable scheduling configuration for a given workload that satisfies system-wide latency, energy, and accuracy requirements.

1.2 Research objectives

The primary objective of this dissertation is to enable efficient runtime orchestration of AI inference across heterogeneous, distributed edge platforms that handle stochastic workload variations. This research aims to design adaptive, energy-efficient, and scalable inference frameworks that utilize compute resources effectively under dynamic workloads. Specific research objectives are defined as follows:

1. **Characterizing and executing diverse AI workloads.** Characterizing and establishing unified execution pipelines that accommodate contemporary edge workloads, ranging from lightweight primitive ML kernels to advanced DNNs, transformers, and LLMs, while jointly optimizing concurrent inference requests across multiple models with architectural diversity and computational heterogeneity.

2. **Leveraging architectural heterogeneity of edge platforms for diverse workloads.** Investigating performance and power characteristics of diverse edge platforms for AI inference with (i) asymmetric CPU clusters, (ii) CPU and GPU clusters, and (iii) CPU and GPU clusters supported by accelerators. Optimizing the resource utilization of heterogeneous computational clusters to

match the latency requirements of the workloads while minimizing the energy consumption.

3. **Designing run-time decision frameworks for meeting the inference requirements.** Developing adaptive strategies that navigate a complex design space at run-time, configuring accuracy, cluster-wide frequency, heterogeneity-aware cluster selection, and priority-based inference execution. Enabling the run-time framework through heuristics or RL support to meet the system-wide requirements, including application latency, accuracy, and energy consumption.

4. **Fine-grained workload partitioning for scalable multi-node execution.** Formulating data and functional partitioning mechanisms that enable flexible distribution of workloads across heterogeneous clusters. Implementing strategies for hierarchical partitioning that support both on-device and edge-cluster workload distribution.

The above objectives collectively address the challenges of executing computationally intensive AI workloads on heterogeneous edge platforms, aiming to bridge the gap between hardware diversity and the stringent latency, power, and accuracy demands of modern AI systems.

1.3 Contributions

Figure 2 provides an overview of the thesis's contributions to on-device inference strategies. It categorizes the research outputs into on-device and distributed inference frameworks, with each contribution linked to its corresponding publication. This mapping highlights how each work tackles specific challenges within different system components. Several novel contributions toward the efficient execution of inference workloads on heterogeneous edge platforms can be summarized as follows:

1. **Joint knob actuation for edge AI orchestration:** This dissertation presents a comprehensive control mechanism for run-time resource management in heterogeneous, multi-core mobile systems. The mechanism combines power, performance, and approximation knobs to ensure coordination between them in a resource-constrained edge environment.

2. **Multi-DNN inference using reinforcement learning:** This dissertation presents multi-DNN orchestration frameworks, *Tango* and *TangoX*, targeting edge platforms with CPU- and GPU-based clusters. The frameworks use PPO-based reinforcement learning to jointly orchestrate application priority, cluster selection, model accuracy, and DVFS states.

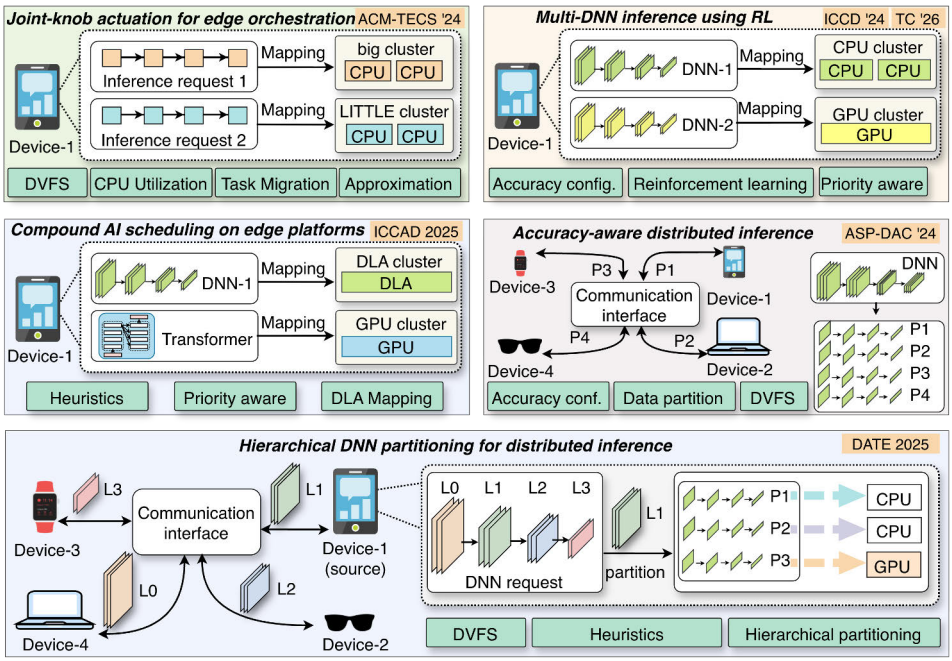

Figure 2. The overview of the thesis contributions and the topics covered in the papers.

3. **Compound AI scheduling on edge platforms:** This dissertation presents a compound AI inference framework *Twill* that jointly orchestrates DNNs, encoder transformers, and Large Language Models (LLMs). *Twill* integrates ONNX-based model profiling with heuristics-based design space exploration to enable AI orchestration across CPU, GPU, and DLA clusters.

4. **Distributed inference through accuracy-aware data partitioning:** This dissertation presents a distributed DNN inference framework for heterogeneous edge clusters with asymmetric CPU clusters. The framework is based on a proportional DNN distribution strategy that jointly optimizes accuracy and latency by partitioning data and approximating streaming image-based workloads.

5. **Hierarchical DNN partitioning framework:** This dissertation presents a hybrid DNN distribution framework, *HiDP*, for deployment on heterogeneous edge nodes with CPU- and GPU-based clusters. *HiDP* is a two-tier model and data-partitioning architecture that jointly configures workload distribution at the global edge-cluster and local node-cluster levels.

Research foundations
The research methodologies and design frameworks developed in this dissertation are guided by a set of research foundations that define the overall philosophy of edge AI orchestration. These principles ensure consistency, generality, and adaptability across all proposed solutions.

1. **On-device heterogeneity:** The AI orchestration strategies are designed and evaluated on heterogeneous edge platforms comprising asymmetric computational clusters, including CPU, GPU, and DLA, to ensure scalability of the methods on a wide variety of modern edge platforms.

2. **Run-time adaptation:** The presented AI inference methods are designed to handle run-time workload and resource variations to sustain real-time latency, energy, and accuracy optimizations.

3. **Scalability to handle diverse AI workloads:** The proposed methods present comprehensive strategies that can be scaled to orchestrate ML workloads with varied architectural complexities.

4. **Disciplined accuracy tuning:** The approximation and partitioning techniques are designed to maintain and ensure user-acceptable model accuracy levels by tuning the scheduling decisions based on the accuracy requirements.

5. **Autonomous decision-making:** The methods are based on a diverse set of design-space exploration strategies, enabling fully autonomous, runtime scheduling decisions with minimal design-time profiling of the workloads.

Together, these foundations define a coherent design philosophy that evolves from localized run-time resource management to distributed, hierarchical orchestration, targeting efficient, intelligent, and scalable AI inference at the edge.

1.4 Organization

This dissertation is a compilation of articles originally published in international conference proceedings and peer-reviewed journal series during the course of this research. The manuscript is organized into two major parts: Part I—Research Summary and Part II—Original Publications.

Part I – Research summary Part I presents a consolidated overview of the research and is structured into chapters 2–7. It provides the theoretical background, motivation, research objectives, methodologies, and key findings of this work. Each chapter discusses a particular aspect of the research progression. Chapter 2 focuses on run-time resource management of heterogeneous multi-core processors. Chapter 3

introduces the *Tango* and *TangoX* frameworks for multi-DNN inference on heterogeneous edge platforms. Chapter 4 extends the exploration to compound AI workloads that combine multiple models, such as vision, transformer, and language models. Chapter 5 presents distributed DNN inference strategies based on data partitioning across collaborative edge devices. Chapter 6 further advances the distributed inference paradigm by introducing hierarchical DNN partitioning. Finally, Chapter 7 concludes the dissertation by summarizing the key findings of the proposed frameworks. The summary of each chapter is shown in Table 1. These chapters provide a unified framework for efficient AI inference across both standalone and collaborative heterogeneous environments, enabling a scalable foundation for next-generation edge intelligence.

Table 1. Summary of chapters and their primary focus areas

Chapter	Focus Area	Key Insights and Results
2	Run-time resource management	Approximation-aware actuation knobs, achieving power-performance coordination on edge platforms.
3	Multi-DNN inference on heterogeneous devices	RL-based scheduling for scheduling concurrent DNN workloads with diverse QoS requirements.
4	Compound AI inference on edge devices	Run-time scheduling of compound AI workloads based on DNN, transformer, and LLMs on edge.
5	Distributed DNN inference with data partitioning	Device heterogeneity aware distributed DNN scheduling using data partitioning and accuracy configuration.
6	Distributed DNN inference with hierarchical partitioning	Scheduling distributed DNN inference by jointly configuring model and data partitioning at global and local levels across an edge cluster.

This dissertation evaluates each strategy as it progressed throughout the research period by comparing all proposed frameworks. The presented research is scalable and builds on prior work in system intelligence, platform enhancements, workload diversity, and logical improvements. Comparative analysis of each chapter is given in Table 2. The steady improvement across frameworks demonstrates a progression from static single-node optimization to dynamic distributed orchestration. The transition from the *RTM* to *HiDP* framework reflects scalability in platforms, workloads, and scheduling complexity. Each design iteration incorporated more autonomy and awareness, effectively unifying partitioning strategies.

Part II – Original publications Part II includes Papers I–VI, which constitute the original publications forming the core of this dissertation. Each paper aligns with a particular section of Part I, as detailed below:

- **Paper I** corresponds to the content presented in **Chapter 2**.
- **Papers II** and **III** correspond to the content presented in **Chapter 3**.
- **Paper IV** corresponds to the content presented in **Chapter 4**.

Table 2. Comparative analysis of proposed frameworks across chapters

Framework	**Scope**	**Platform**	**Optimization domain**	**Design space exploration**
RTM (Ch.2)	Multi-app execution	Single-device (CPU only)	Power, performance, accuracy	Heuristics
TangoX (Ch.3)	Multi-DNN scheduling	Single-device (CPU, GPU)	Application priority, latency, energy	PPO-RL
Twill (Ch.4)	Compound AI workloads	Single-device (CPU, GPU, DLA)	Application priority, latency	Heuristics
Proportional (Ch.5)	Data partitioning on distributed network	Multi-device (CPU-only)	Accuracy, performance	Dynamic programming
HiDP (Ch.6)	Hierarchical partitioning on distributed network	Multi-device (CPU, GPU)	latency	Heuristics

- **Paper V** corresponds to the content presented in **Chapter 5**.
- **Paper VI** corresponds to the content presented in **Chapter 6**.

Each of these publications expands upon the individual methodologies, experiments, and results that collectively constitute the overarching research presented in Part I.

2 Run-time Resource Management of Heterogeneous Multi-core Processors

Heterogeneous Multi-core Processors (HMPs) feature asymmetric computational clusters with diverse power and performance characteristics [25; 26]. Mobile and edge Heterogeneous Multi-core Processors (HMPs) operate under strict power budgets and have limited performance. Modern AI applications require inference of diverse ML models on HMPs. These workloads typically employ dense network architectures that involve computationally intensive Multiply–Accumulate operation (MAC) operations. Inference requests arrive stochastically at the system, necessitating dynamic performance and accuracy requirements. Therefore, HMPs require an intelligent resource management framework to meet the requirements of compute-intensive AI workloads. HMPs support multiple system-level actuation knobs to optimize the power consumption and performance. These knobs allow adjusting cluster-wide operating frequencies, application-to-cluster mapping, and per-core utilization levels [9; 8]. The workload actuation knobs adjust the degree of parallelism and manage accuracy–performance trade-offs through run-time approximation. Approximation can provide performance gain for ML applications that show error-resilience and allow accuracy-performance trade-off [27]. Efficient execution of inference workloads on HMPs requires intelligent Run-time Resource Management (RTM) strategies that observe and adapt system- and workload-level parameters to jointly optimize power, performance, and accuracy.

This chapter presents an accuracy-aware run-time management framework that augments conventional power-control mechanisms with run-time approximation capabilities to optimize the workload's performance within the available power budget. The framework efficiently balances power and performance for multi-programmed and multi-threaded workloads on mobile HMPs by enabling informed control decisions under varying operating conditions. The details of the proposed framework were initially published in the following article:

I **Zain Taufique**, Anil Kanduri, Antonio Miele, Amir Rahmani, Cristiana Bolchini, Nikil Dutt, Pasi Liljeberg. *Exploiting Approximation for Run-time Resource Management of Embedded HMPs*. ACM Transactions on Embedded Computing Systems (ACM-TECS), 2024.

The key contributions to the proposed framework include:

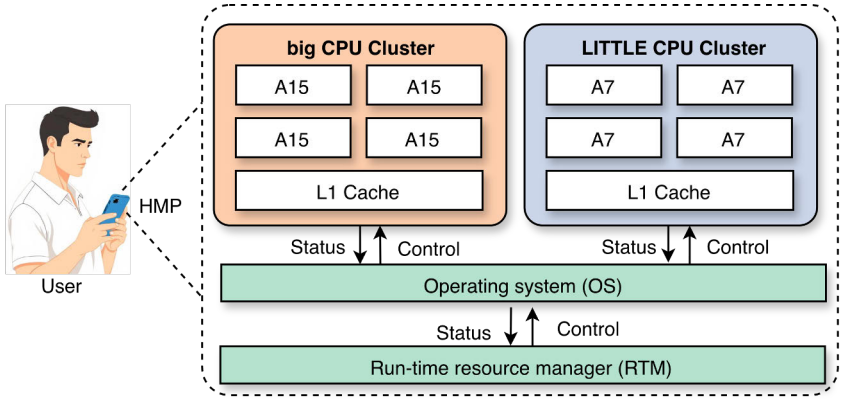

Figure 3. Samsung Exynos 5422 System-on-Chip heterogeneous multi-core platform

- An RTM framework in mobile HMP systems, designed to cater unpredictable and dynamic workload variations arising from concurrent workload execution.
- Joint orchestration of approximation, application-to-core mapping, Dynamic Voltage/Frequency Scaling (DVFS), *CPU Quota* settings, Task Migration (TM), and Degree of Parallelism (DoP), while accounting for performance goals and power constraints.
- Implementing analytical power and performance estimation models for enabling intelligent resource management.

2.1 Need for run-time resource management on HMPs

A prominent architecture type for HMP includes the ARM *big.LITTLE* comprising high-performance *big* cores with energy-efficient *LITTLE* cores, providing opportunities to orchestrate diverse AI workloads [22; 28; 29]. Figure 3 illustrates the popular Samsung Exynos 5422 architecture, which features four ARM A7 CPU cores and four ARM A15 cores, comprising the *LITTLE* and *big* clusters, respectively. The architecture is typically supported by an Operating System (OS) that provides an interface to RTM strategies for monitoring and controlling workload execution.

By default, Linux governors, such as ondemand, performance, and powersave, perform RTM for a given workload. These governors rely on simple cluster mapping and threshold-based scaling of cluster frequencies. Linux governors react to coarse utilization levels rather than predictive or fine-grained workload characteristics [30; 23]. Therefore, the default Linux governors typically fail to meet the system-wide power and performance requirements [31]. State-of-the-Art (SoA) RTM strategies primarily rely on actuating only system-wide control knobs to meet performance objectives under a fixed power budget [9; 8; 32; 33]. These approaches are effec-

tive in controlled or single-application settings but exhibit limited efficiency under unknown and dynamic workload variations. Such variations include stochastic application arrival and exit times, changes in execution behavior due to performance requirements, and dynamic performance requirements across concurrently running workloads. In particular, they lack coordinated power–performance reasoning across concurrently executing applications. Power-actuation decisions for one application implicitly affect the performance and power headroom of other applications sharing the same cluster. As a result, these strategies often incur performance degradation during aggressive power actuation or violate power budgets when provisioning performance under dynamic conditions.

2.2 Actuation knob dynamics

The effectiveness of RTM strategies on HMPs is fundamentally determined by the set of system-level and workload-level actuation knobs. These knobs allow the RTM strategies to coordinate power and performance by selectively scaling the available resources [34]. Traditional system-level knobs actuate application-to-core mapping, task migration, DVFS, and CPU quota assignment. Advanced workload actuation knobs include DoP and approximation. Figure 4 illustrates the impact of actuating DVFS, CPU quota, DoP, and approximation on the average workload performance mapped to the *big* and *LITTLE* clusters of the Odroid XU3 platform [35]. The relative performance gain differs across knobs, reflecting their distinct actuation semantics. Frequency scaling yields a significant cluster-wide performance gain at the cost of high power consumption. CPU quota and approximation enable fine-grained performance modulation with fundamentally different power and accuracy implications. Each knob differs in granularity and scope, complicating the joint knob actuation scenarios. As a result, isolated actuation of traditional knobs can become infeasible or suboptimal in multi-programmed scenarios, particularly when performance recovery for one workload degrades co-located workloads or violates global power constraints. This dissertation proposes that intelligent RTM strategies should consider the impact of jointly actuating multiple knobs across concurrently running applications in clusters. However, the exploration space is vast and complex, making it difficult to find suitable actuation settings for multiple knobs across multiple applications on heterogeneous clusters. Figure 5 illustrates the actuation settings of different knobs on App-1, mapped to an HMP based on *big.LITTLE* architecture. The left side of each subplot, highlighted in red, depicts the scenarios before the knob actuation is enforced. The right-hand side presents the post-actuation phase, showing how each knob affects the execution of the workload. This section presents the details of workload and system-wide knobs.

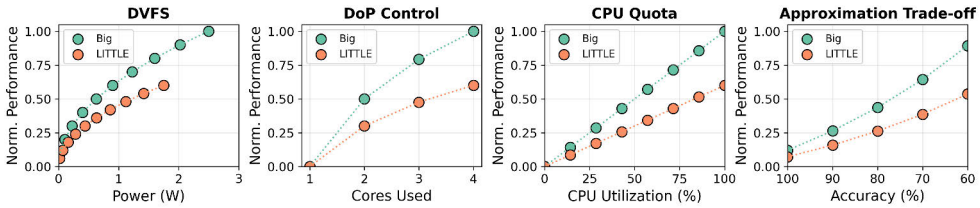

Figure 4. Performance of actuation knobs, including (a). DVFS, (b). DoP, (c). CPU quota, (d). Approximation

2.2.1 System-level knobs

Application-to-core mapping strategies Application-to-core mapping determines which cluster an application/thread run on, directly influencing achievable performance and power consumption. Design-time mapping decisions enable predictable resource allocation but cannot respond to workload variation or phase changes, often leading to underutilization or performance degradation under dynamic conditions [9]. Run-time mapping decision leverages resource monitoring to migrate performance-critical or compute-intensive applications to *big* cores, while placing latency-tolerant or background workloads on *LITTLE* cores [36]. Since the cluster-level resources are shared, mapping decisions for one application can alter the execution conditions of co-located workloads, introducing indirect performance effects through shared contention [37]. Consequently, the suitability of executing compute-intensive workloads on HMPs must be continuously re-evaluated against the resulting performance degradation and power savings.

Task migration across clusters Task migration (TM) between the *big* and *LITTLE* clusters exploits the architectural heterogeneity on HMPs. Figure 5(b) illustrates the migration of application-1 from the *LITTLE* cluster to the *big* cluster to improve performance. Effective RTM strategies must evaluate migration decisions in the context of application performance requirements, power consumption, and the availability of idle cores on the destination cluster. In multi-programmed scenarios where both clusters are busy, migration decisions are tightly coupled with the execution of concurrently running applications. Running compute-intensive workloads on the *LITTLE* cluster can cause significant performance degradation, which must be mitigated by the joint actuation of a subsequent knob. Intelligent RTM strategies employ heuristics such as performance-loss thresholds and historical profiling to estimate the performance impact before enforcement [38].

CPU utilization control and run-time adaptation CPU quota assignment is a process-level control mechanism exposed through OS-level interfaces such as CGroups [39]. Figure 5 (d) shows the actuation of the CPU quota, where the CPU utilization of App-1 is reducing from 100% to 80% on the *big* cluster. By regulating the upper bound on the number of CPU cycles allocated to a process, the system can

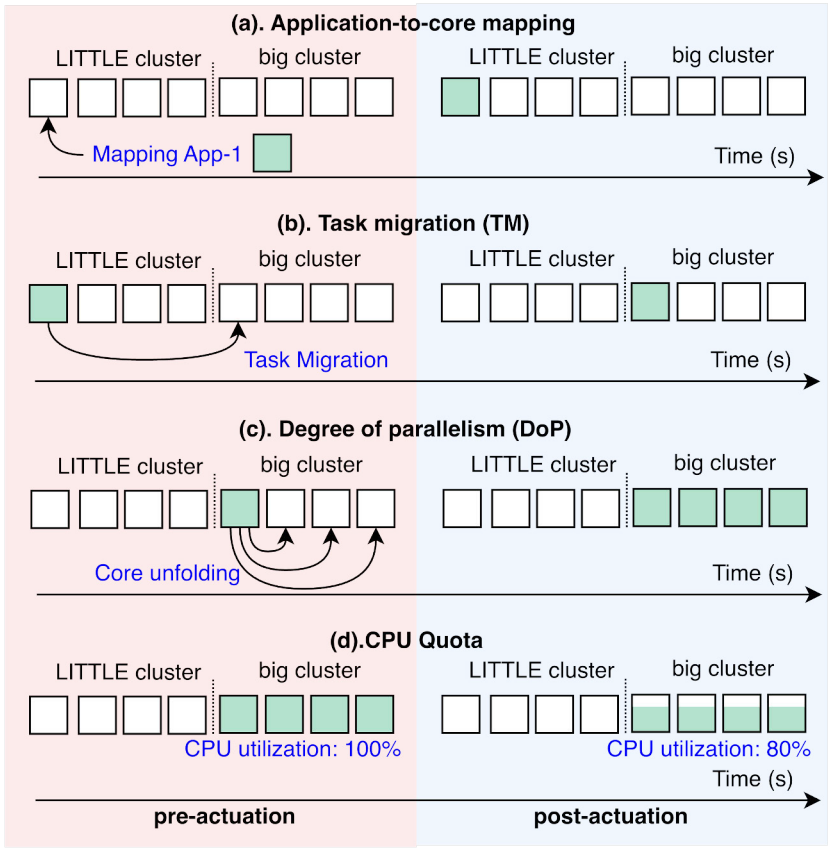

Figure 5. Actuation knobs applied on application-1 by controlling (a). Application-core-mapping, (b). Task migration (c). Degree of Parallelism (DoP), and (d). CPU quota

modulate application throughput without altering frequency or mapping [34]. Unlike cluster-wide DVFS, CPU quota provides per-application granularity, enabling targeted slowdowns or accelerations without affecting other applications. For streaming ML kernels, proportional adjustments to the CPU quota often result in nearly linear changes in throughput [40]. However, quota enforcement must balance fairness and performance requirements, especially when multiple applications exhibit dynamic arrival patterns and variable processing intensity.

Dynamic voltage and frequency scaling Dynamic voltage and frequency scaling (DVFS) is a widely used mechanism for balancing computational performance with power savings. Frequency selection is typically accompanied by proportional voltage scaling, which impacts power consumption quadratically [28]. Heterogeneous platforms implement DVFS per cluster, meaning that adjusting the frequency of big cores affects all cluster-wide running tasks. This cluster-level granularity introduces interference among tasks, limiting control over power and performance coordina-

tion. Under variable workloads, reactive frequency governors, such as *Ondemand* and *Interactive*, scale the CPU frequency based solely on recent utilization statistics [30; 8]. While effective in single-cluster or homogeneous settings, these governors may either overcompensate or undercompensate for performance demands in heterogeneous multi-application scenarios, potentially resulting in Quality of Service (QoS) violations [41]. Consequently, DVFS decisions must account for both real-time performance requirements and the impact on concurrently executing applications, particularly when operating under strict thermal or power-envelope constraints.

2.2.2 Application-level knobs

Approximation Approximation serves as a higher-level actuation knob that leverages tolerance to inexact computation present in many ML workloads. Approximate computing techniques adjust algorithmic precision, skip iterations, or use simplified models when accuracy loss remains acceptable to the application or user [42; 43]. Approximation can prevent performance degradation when system knobs cannot meet workload requirements within power budgets. For tasks such as clustering or regression, modest accuracy loss enables significant power savings and improved throughput. However, trade-offs vary across workloads, and indiscriminate use of approximation can introduce instability or unpredictable behavior. Therefore, runtime systems typically monitor performance trends and apply approximations only when traditional knobs reach their limitations [34]. In multi-application environments, selectively approximating the most error-resilient task can maintain global performance without compromising excessive accuracy, offering a complementary tool alongside hardware-level controls.

Degree-of-Parallelism control for applications DoP specifies the number of cores allocated to an application. DoP control is application-dependent and feasible only for parallel applications that support run-time workload splitting for simultaneous execution across multiple cores. Fine-grained DoP regulation provides flexibility to tailor resource allocation, particularly in edge environments where multiple parallel kernels frequently coexist [44]. Figure 5 (c) shows that the number of cores allocated to App-1 increases to 4, occupying all available cores on the *big* cluster. Increasing and decreasing DoP affect the application's performance and power consumption [9]. DoP control therefore requires awareness of application scalability characteristics, cluster capacity, and power constraints. Under resource constraints, allocating more cores to one workload affects the performance of another application running on the same cluster.

2.3 Existing strategies for RTM on mobile HMPs

RTM for embedded and edge HMP-System-on-Chips (SoCs) primarily focuses on balancing performance and power under tight Thermal Design Power (TDP) and performance constraints. Classical approaches rely on traditional power/performance knobs such as DVFS, TM, *CPU quota*, application mapping, and DoP to maintain desired performance while reducing power consumption [28; 45; 44; 41]. Linux governors widely deployed on mobile SoCs (e.g., *ondemand*, *interactive*) are effective for stable single-objective scenarios. Yet, they often fail to achieve the required performance when system and workload dynamics change rapidly. Advanced ML-guided RTM mechanisms leverage offline profiling, online models, or hybrid schemes to optimize power-performance trade-offs [33; 8; 9; 32; 46; 38]. However, under concurrent execution of unknown workloads, these strategies can still exhibit power overshoots, QoS violations, or sub-optimal power utilization. These strategies are unable to coordinate among available actuation knobs and rely solely on static learned behaviors.

Approximate computing exploits the inherent error resilience of many AI/ML and signal-processing workloads to enhance throughput and energy efficiency [47]. Prior studies combine approximation with classical knobs to mitigate performance degradation caused by power throttling in heterogeneous embedded systems [34]. Early strategies introduce approximation as a reactive knob for many-core and HMP architectures [42; 43], switching between accurate and approximated modes when performance falls below set thresholds. These approaches can, however, trigger conflicting power-performance actions and oscillatory behavior due to uncoordinated decision-making. Other works apply approximation in constrained contexts such as thermal control or video applications [48; 49; 50], but rely on coarse design-time approximation granularity, thereby limiting the exploitable accuracy-performance-power search space. Furthermore, many approximation-aware RTM techniques assume single-application or static workloads, ignoring heterogeneous multi-programmed scenarios. As a result, they lack mechanisms to prioritize more error-tolerant tasks, leading to either unnecessary degradation of accuracy or limited benefits when workloads vary widely. In summary, existing RTM and approximation-enabled techniques do not adequately coordinate power knobs with accuracy trade-offs, nor do they account for diverse dynamic workloads, thereby constraining their ability to sustain QoS with optimal Performance per Watt (PPW) under tight energy budgets.

2.4 Proposed RTM framework for asymmetric HMPs

This dissertation presents a modular RTM framework that operates as a closed-loop control system across the application, OS, and hardware layers as shown in Fig-

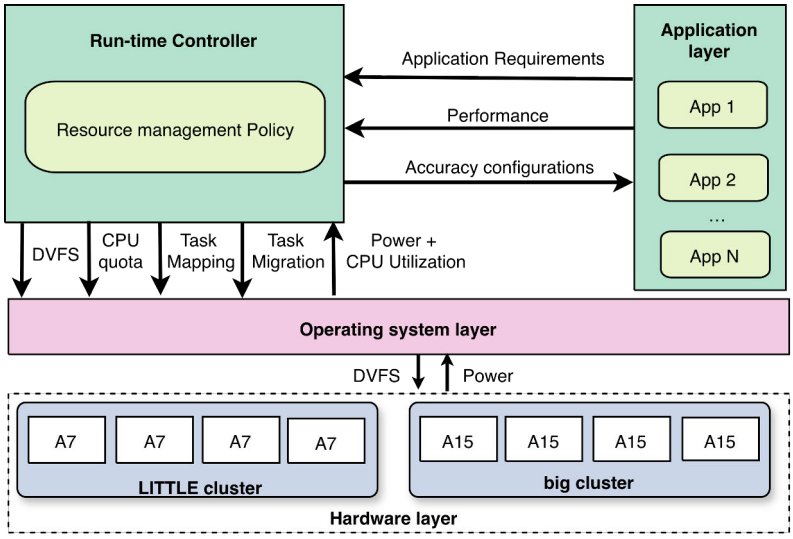

Figure 6. Proposed framework for run-time resource management on HMPs

ure 6. The run-time controller sits above the OS and hosts the resource management policy, which decides the optimal configuration of execution knobs such as DVFS, TM, DoP, CPU quotas, and application-to-core mapping. Applications (App 1 to App N) execute in the application layer and may arrive dynamically, exposing performance targets and, when applicable, supporting accuracy adaptation. The controller continuously monitors performance and power behavior, evaluates available power headroom, and determines knob actuation settings accordingly. These decisions are enforced through OS interfaces that allow applications to be bound to specific compute clusters, regulate CPU time slices, adjust frequencies, and steer execution between *big* and *LITTLE* cores in the underlying HMP SoCs. The hardware layer provides heterogeneous compute resources with onboard power monitoring, enabling the controller to meet TDP constraints while maintaining QoS. Overall, the framework operates in a feedback loop, where the applications generate workload demands, the controller observes system status, selects knob actuations, and the OS executes those commands, ensuring that performance requirements are met.

2.5 Experimental setup and evaluation

The proposed RTM framework is implemented in C++ as a modular middleware comprising a run-time controller, monitoring units, a resource allocation policy, and an OS interface. It runs as a user-space process in a Linux environment, using native drivers for power sensing and DVFS control, and the CGroups library for task-to-core mapping and *CPU Quota* management. Applications are instrumented with the *HeartBeat* library to express high-level performance requirements and monitor

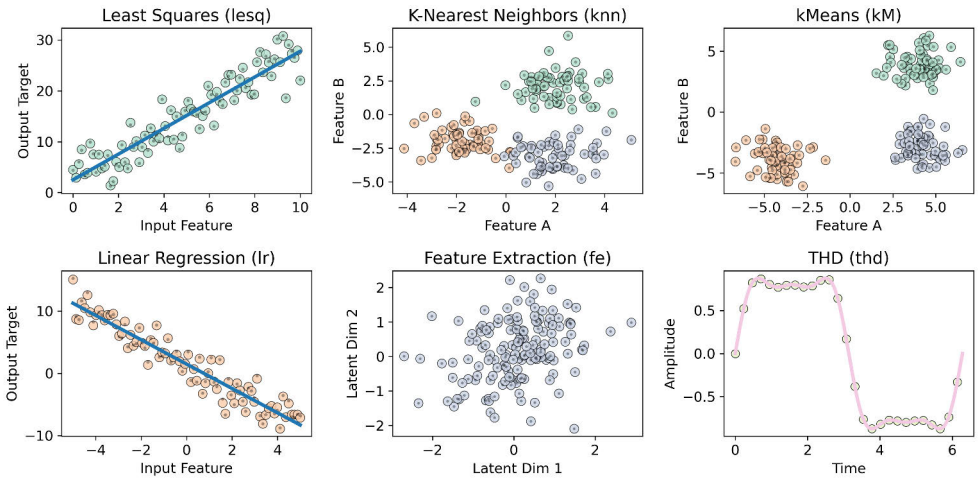

Figure 7. Benchmark workloads spanning regression, classification, clustering, feature extraction, and signal analysis. The set includes (a) least squares, (b) K-nearest neighbors, (c) k-means, (d) linear regression, (e) latent-space feature extraction, and (f) total harmonic distortion.

execution. The controller executes periodically with a 1s *control period*. The proposed RTM strategy is platform-independent and requires only minor adjustments to device file paths when ported to other platforms. Experiments were conducted on two embedded platforms, the Odroid XU3 [35] and the Tinker Edge R [51], both of which expose core-level heterogeneity and distinct power and performance characteristics. The Odroid XU3 integrates Samsung's Exynos 5422 SoC featuring an ARM *big.LITTLE* processor. The Tinker Edge R hosts a Rockchip RK3399Pro SoC with two A72 (*big*) and four A53 (*LITTLE*) cores.

The proposed framework was evaluated through different ML applications as shown in Figure 7, including Least squares, K-Nearest Neighbors (KNN), k-Means (kM), Linear regression, Feature Extraction (FE), and Total Harmonic Distortion (THD). These applications support diverse use cases for streaming input data. The benchmark applications, shown in Figure 7, cover regression, classification, clustering, feature extraction, and signal analysis tasks. Each workload exhibits distinct computational structures, data access patterns, and sensitivity to approximation, reflecting heterogeneous performance, accuracy, and power behaviors. Together, these benchmarks capture a representative cross-section of machine learning kernels commonly executed on edge and embedded platforms, enabling systematic evaluation of run-time resource management strategies.

Least Squares minimizes squared error over 1M samples using loop perforation for configurable approximation [52]. Common in embedded estimation tasks, it provides good error resilience but exhibits iteration-dependent variability, requiring coordinated knob control to maintain stable performance [47].

K Nearest Neighbors (KNN) implements a memory-intensive search stage from *mlpack* [52]. Used in lightweight edge ML, its performance is sensitive to data distribution and bandwidth constraints, often requiring careful mapping and DVFS tuning under multi-programmed execution.
kMeans from Rodinia [53] clusters 50k points with relaxation-enabled convergence. It is widely used in unsupervised learning, but its per-iteration costs fluctuate, straining computational and memory resources. Approximation accelerates convergence, though excessive relaxation can degrade cluster quality.
Linear Regression (Linear Regression (LR)) operates as a serial regression kernel on 1M samples [53]. While lightweight, LR has limited tolerance to approximation; even a small number of loop-skipped iterations can cause noticeable accuracy loss [47], necessitating conservative tuning when co-scheduled with heavier workloads.
Feature Extraction (FE) represents front-end signal-processing stages in Internet of Things (IoT) and vision pipelines [54]. Its computational and memory footprints vary with input characteristics, making it sensitive to runtime fluctuations and shared-resource contention.
Total Harmonic Distortion (THD) computes harmonic distortion for sensing and power-quality monitoring [55]. As a parallel, throughput-intensive kernel, THD is challenging under tight power budgets, often requiring strict, latency-aware resource allocation on embedded SoCs.

The workload performance is profiled at 5-s intervals during run-time, and offline-generated lookup tables capture the approximation–performance trade-off. Throughput requirements are expressed as a range between TP_{min} and TP_{max}, determined from standalone runs at 80% and 100% of maximum frequency, respectively. The proposed approach is compared with two SoA HMP management strategies, *AdaMD* [9] and *DAgger* [8], both relying on offline profiling and machine learning models to infer optimal DVFS and DoP configurations. For completeness, the proposed strategy is benchmarked against Linux's *Ondemand* (*HMP_O*) and *Interactive* (*HMP_I*) governors, which adjust CPU frequency based on the workload intensity.
Evaluation summary The proposed RTM strategy was evaluated under two workload scenarios, executing combinations of two and three concurrent applications drawn from the considered workload set. Table 3 reports the average power and performance violation rates observed across all experiments on the Odroid XU-3 and Tinker Edge R platforms for the five evaluated RTM strategies. Across both platforms and workload intensities, the proposed strategy avoids power and performance violations while incurring only limited accuracy loss when approximation is explicitly invoked.

Figure 8 aggregates these results by illustrating the average power violation, per-application performance violation, and accuracy loss across all experiments. On the Odroid XU-3 platform (Figure 8(a)), the standard Linux governors (HMP_O

Table 3. Power and performance violation rates when running two and three applications.

Platform	Scenario	Strategy	Power Violation (%)	Performance Violation (%)
Odroid XU-3	2 apps	AdaMD	1.5	34
		DAgger	0.3	28
		HMP_O	13.0	100
		HMP_I	11.5	90
		Ours	**0.0**	**0**
	3 apps	AdaMD	2.2	48
		DAgger	0.6	35
		HMP_O	12.8	100
		HMP_I	11.2	95
		Ours	**0.0**	**0**
Tinker Edge R	2 apps	AdaMD	1.0	22
		DAgger	0.15	18
		HMP_O	0.0	98
		HMP_I	0.0	95
		Ours	**0.0**	**0**
	3 apps	AdaMD	1.1	27
		DAgger	0.2	21
		HMP_O	0.0	97
		HMP_I	0.0	93
		Ours	**0.0**	**0**

and HMP_I) exhibit substantial power violations, exceeding 11–13%, while simultaneously suffering from severe performance violations ranging between 90–100% across multiple applications. These results indicate that reactive frequency scaling alone is insufficient to manage concurrent workloads under a strict power budget. AdaMD and DAgger reduce power violations to below 2.2% (Table 3, Odroid XU-3), but still incur significant performance violations of 28 to 48%, reflecting limited coordination between power control and per-application performance management. A similar behavior is observed on the Tinker Edge R platform (Figure 8(b)). Although overall power violations remain low for all strategies (below 1.1%, as shown in Table 3), AdaMD and DAgger continue to exhibit performance violations between 18–27% when running two and three concurrent applications. The standard Linux governors again fail to meet application performance requirements, with performance violations exceeding 90% across several workloads due to aggressive downscaling during workload variations. In contrast, the proposed strategy consistently maintains both power and performance within the specified constraints across all scenarios.

The accuracy loss observed with the proposed approach is confined to a small subset of compute-intensive applications, as shown in the rightmost panels of Figure 8. The maximum accuracy degradation is limited to 6 to 7% on Odroid XU-3 and approximately 4% on Tinker Edge R, and occurs only when approximation is selectively invoked to compensate for exhausted traditional knobs. This confirms that

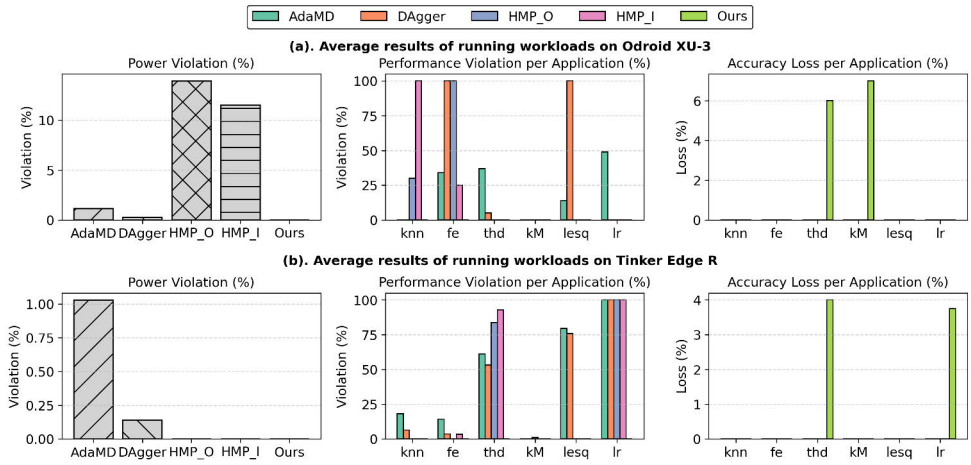

Figure 8. Power and performance violation across all the experiments on platforms (a) Odroid XU-3 and (b) Tinker Edge R

the approximation is applied in a controlled, necessity-driven manner rather than as a default mechanism.

Overall, these results demonstrate that jointly actuating traditional system knobs with application-level approximation enables effective coordination between power and performance objectives under dynamic, multi-programmed workloads. Unlike existing approaches that prioritize either power adherence or performance provisioning in isolation, the proposed strategy consistently satisfies both constraints while bounding accuracy loss, validating its suitability for run-time resource management on heterogeneous edge platforms for ML orchestration.

3 Multi-DNN Inference on Heterogeneous Edge Devices

Inference of DNN workloads on heterogeneous edge devices is crucial for performing multiple cognitive tasks on image-based inputs, including image classification, semantic segmentation, object detection, and scene detection [56]. The DNN models for diverse tasks often exhibit distinct architectures, parameter densities, computational complexities, and performance requirements [57]. Advanced vision applications require simultaneous execution of multiple DNN tasks for different scenarios [12]. Concurrent execution of multiple DNNs introduces contention for shared compute resources, leading to variability in latency and energy consumption [15]. The previous Chapter 2 has discussed the utility of edge platforms with asymmetric CPU clusters for AI workloads. However, modern edge devices, including smartphones, smart glasses, AR gear, and VR gear, are equipped with asymmetric CPUs and high-performance GPU clusters [58]. Managing workload and system dynamics requires scheduling strategies that adapt to stochastic workload arrivals, meet QoS requirements, and account for the heterogeneous energy–latency profiles of asymmetric-CPU and GPU clusters [58]. The previous chapter focused on resource management on heterogeneous edge platforms with only CPU clusters. This Chapter discusses the significance, challenges, and novel solutions for executing multi-DNN workloads on edge devices with heterogeneous CPU-GPU clusters to meet latency, energy, and accuracy requirements. This Chapter is based on contributions that are published in the following articles:

II **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Tango: Low latency Multi-DNN inference on heterogeneous edge platforms*. IEEE 42nd International Conference on Computer Design (ICCD), 2024.

III **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *TangoX: Robust Multi-DNN Inference on Heterogeneous Edge Platforms*. IEEE Transactions on Computers (IEEE-TC), 2026 (Under review).

The contributions include:

- *TangoX*, a resilient multi-DNN inference framework that manages concurrent DNN requests with diverse QoS requirements, including priority levels, latency constraints, and accuracy demands.

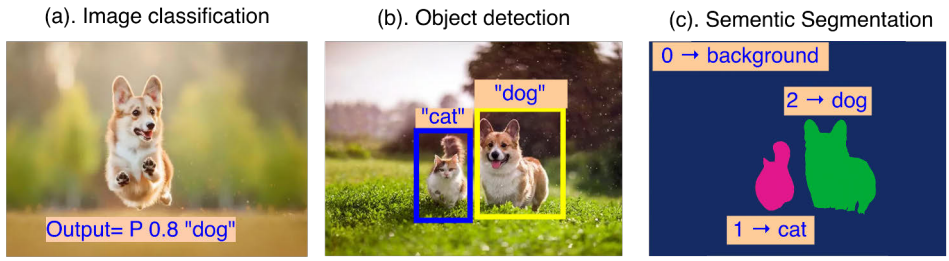

Figure 9. Vision tasks (a). image classification, (b). Object detection, (c). Semantic segmentation

- Accuracy configuration for meeting the QoS requirements through run-time DNN model selection.

- PPO-based reinforcement learning agent to jointly actuate cluster allocation, accuracy configuration, DVFS tuning, inter-cluster workload migration, and task prioritization.

3.1 Significance of multi-DNN inference on edge

Mobile and edge computing applications require inference across multi-DNN workloads to perform simultaneous tasks such as image classification, semantic segmentation, and object detection. The multi-DNN requests can arrive randomly, driven by user prompts or application design, with assigned QoS requirements for execution priorities, latency targets, and accuracy. Applications in a multi-DNN workload can have varying QoS requirements, necessitating run-time orchestration to streamline execution and improve user experience. Multi-DNN inference on resource-constrained, heterogeneous edge devices poses complex challenges, requiring extensive run-time exploration of configurations that satisfy task-specific requirements. The scheduling challenge is intensified by constrained compute capacity, tight energy budgets, and heterogeneous core architectures, including asymmetric CPUs and GPUs. It is further shaped by differences in computational characteristics and latency requirements across multi-DNNs. In addition, stochastic run-time factors, such as cluster contention, cluster availability, and workload variability in dynamic multi-DNN scenarios, further complicate achieving low-latency execution. Addressing inference latency under these conditions necessitates intelligent run-time orchestration that schedules multi-DNN workloads with awareness of system- and workload-level dynamics. This section presents each scheduling challenge in detail.

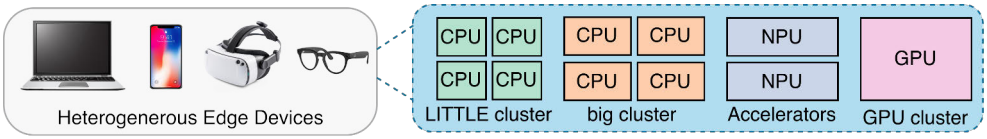

Figure 10. Hardware architecture of heterogeneous edge devices

3.1.1 Handling task-level heterogeneity of multi-DNNs

In multi-DNN inference scenarios, task-level heterogeneity arises from the simultaneous execution of distinct vision tasks, such as image classification, object detection, and semantic segmentation, each performing fundamentally different computational activities as illustrated in Figure 9. Although these tasks operate on the same image-based inputs, they rely on DNNs with markedly different architectures, execution workflows, and computational structures. Image classification typically follows a single-pass feed-forward pipeline optimized for global feature aggregation. Object detection involves multi-stage processing, including region proposal generation and bounding box regression. Semantic segmentation requires dense, pixel-level inference with high-resolution feature maps. These differences result in non-uniform model depths, parameter densities, memory access patterns, and MAC operation distributions, which, in turn, lead to task-specific latency, energy consumption, and resource utilization characteristics at run-time. Consequently, a multi-DNN workload cannot be treated as a homogeneous set of inference requests. Each task imposes distinct, often conflicting demands on the underlying compute resources, thereby exposing task-level heterogeneity as a fundamental challenge in coordinating concurrent DNN inference on edge platforms.

3.1.2 Handling core-level heterogeneity

Modern edge devices feature diverse computational clusters with heterogeneous energy and latency characteristics. Figure 10 illustrates a conceptual architecture of the Jetson series embedded platforms. The computational clusters of a Jetson board include *big.LITTLE* CPU cluster, GPU cluster, and accelerators such as NVIDIA Deep Learning Accelerator (NVDLA). For a given DNN model, each computational cluster exhibits asymmetric energy and performance characteristics. Each cluster is optimized for specific types of compute operations such as convolution, matrix multiplication, or activation functions. The performance and energy diversity across different clusters stem from variations in microarchitectural design, including pipeline depth, memory hierarchy, and functional unit specialization. Run-time scheduling of inference workloads on heterogeneous edge devices, therefore, necessitates aligning workload characteristics with the most suitable computational clusters to ensure efficient execution under varying performance and energy constraints.

3.1.3 Priority-awareness and Quality-of-Service requirements

Mobile AI services commonly involve a combination of continuous streaming and event-driven inference [14]. In general, event-driven inference requests are given higher priority compared to continuous inference tasks [59; 60]. When a low-priority inference task is in progress, the sudden arrival of a high-priority request necessitates the dynamic reconfiguration of run-time resources. Executing a prioritized DNN may require migrating concurrently running DNNs from the GPU to the CPU, thereby degrading performance. Therefore, effective multi-DNN inference strategies must support preemption mechanisms that ensure both task priority and overall performance.

The DNN tasks have diverse QoS requirements, including target latency, accuracy, and priority. Depending on the application scenario and urgency, each task has an associated QoS requirement vector, either defined by the logic designer or by the user [60]. For example, in an autonomous setup, object detection may prioritize obstacle detection due to continuous motion and scene changes. However, for a VR gear setup, semantic segmentation may have higher user priority for creating an immersive environment [61]. Similarly, applications with higher priority may require lower latency and higher accuracy to support streamlined operations. A run-time scheduling of DNN tasks with varying QoS requirements on heterogeneous edge devices is a complex problem.

3.2 Existing multi-DNN orchestration strategies

Existing multi-DNN inference strategies leverage both CPU and GPU resources to optimize one or more QoS objectives through various strategies: pipelining DNN layers across CPU–GPU clusters [22], distributing workloads between them [14; 15], selecting the most suitable cluster for each DNN [12; 62], assigning priorities among concurrently executing models [60], adjusting DVFS levels [12; 63; 64], and dynamically configuring model accuracy [65; 66]. Current multi-DNN inference scheduling methods typically rely on offline profiling to partition the DNNs into multiple-layer blocks, which are then assigned to different processing clusters for coordinated execution [22; 12]. These techniques typically assume a static inference workload, in which the scheduler has prior knowledge of request arrivals at design time. In contrast, mobile AI workloads exhibit *stochastic run-time variance*, characterized by unpredictable fluctuations in workload intensity, unknown arrival or departure times of inference requests, and changing task priorities and performance demands. Such variability impacts both resource availability and system performance targets, necessitating dynamic reallocation of computational resources based on QoS requirements, cluster status, and DNN-specific performance characteristics. Because existing multi-DNN scheduling schemes overlook run-time variability, their effectiveness

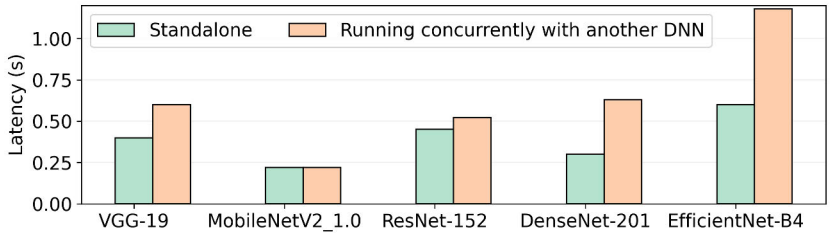

Figure 11. Latency comparison of running a DNN model standalone and simultaneously with another DNN on GPU

is limited to static conditions—often at the expense of maintaining desired QoS.

Existing methods primarily focus on optimizing global throughput, enabling low-latency inference for DNNs deployed on the GPU, but often at the expense of increased latency for concurrent DNNs running on the CPU. Other studies [14; 62] have integrated dynamic voltage and frequency scaling (DVFS) into coordinated DNN scheduling to further enhance performance. However, due to the strict power constraints of edge devices, aggressive frequency scaling offers only marginal benefits. Consequently, existing multi-DNN inference approaches often fail to meet the latency requirements of co-located DNNs, underscoring the need for alternative low-latency scheduling techniques. This highlights the need for adaptive run-time scheduling frameworks capable of handling diverse, dynamic workloads.

3.3 Orchestrating multi-DNN inference on CPU-GPU clusters

Under multi-DNN workloads, the inference latency of a DNN is influenced by shared resource contention among concurrently executing DNNs. This challenge is exacerbated by: (i) limited compute capacity, restricted energy budgets, and heterogeneous core architectures (e.g., asymmetric CPUs, GPUs. This section discusses the challenges of resource contention and the existing frameworks for multi-DNN orchestration on the edge.

3.3.1 Resource congestion on GPU with multi-DNN workloads

Figure 11 presents the inference latency of five DNNs when executed in (i) standalone mode (a single DNN) and (ii) multi-DNN mode (simultaneously with other DNNs) on GPU clusters. The multi-DNN workload configurations include VGG-19 + MobileNetV2_1.0, MobileNetV2_1.0 + ResNet-152, ResNet-152 + DenseNet-201, DenseNet-201 + EfficientNet-B4, and EfficientNet-B4 + VGG-19. By default, standard Linux governors prioritize GPU allocation for DNN applications running in standalone mode. As depicted in Figure 11, the inference latency of each DNN

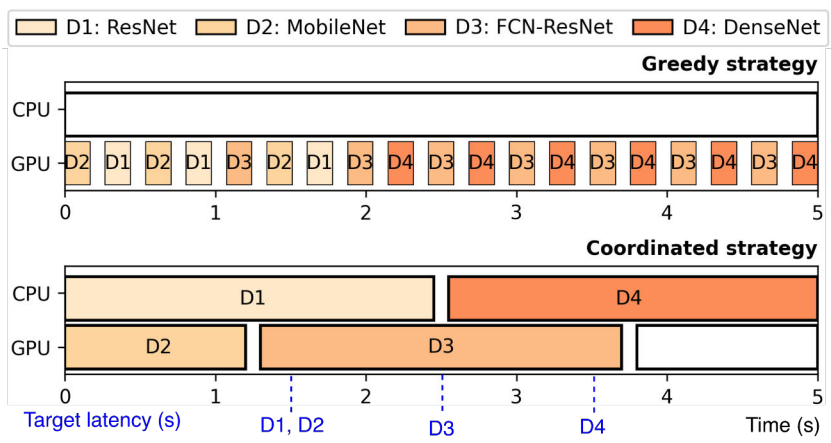

Figure 12. The latency of DNNs running standalone and concurrent with another DNN on GPU

increases when executed concurrently under multi-DNN workloads. For instance, VGG-19 exhibits an inference latency of 0.4s on the GPU in standalone mode, which rises to 0.6s when MobileNetV2 executes concurrently on the same GPU. The inference latency of a DNN varies across different multi-DNN configurations due to the models' diverse computational characteristics. Furthermore, the practical impact of this latency penalty depends on the tolerance for latency of a given inference request. For example, a VGG-19 request with a 3s latency target remains unaffected by the VGG-19 + MobileNetV2 combination, whereas a 0.5s latency target would be violated under the same multi-DNN setup. Hence, multi-DNN inference strategies should incorporate interference awareness and follow an interference-minimization objective to determine scheduling configurations on edge devices.

3.3.2 Conservative multi-DNN orchestration frameworks

To illustrate the performance of various multi-DNN inference strategies, an example workload containing inference requests for four models is evaluated, including *D1: ResNet* [67], *D2: MobileNet* [68], *D3: FCN-ResNet* [69], and *D4: DenseNet* [70]. This scenario assumes that the last image classification application, D4, is a user-triggered process with the highest priority. Figure 12 shows how different scheduling approaches—namely *Greedy* and *Coordinated*—influence run-time behavior. In this setup, inference requests for D1 and D2 arrive concurrently at $t = 0$s, followed by D3 at $t = 1$s and D4 at $t = 2$s. Each application requires inference on a 32-image batch with a target latency of 1500ms. The *Greedy* approach, analogous to default Linux governors, assigns all incoming DNNs to the GPU and tunes DVFS based on workload intensity. This results in time-shared execution across DNNs, leading to frequent context switches and elevated inference latency. On the other hand, *Co-*

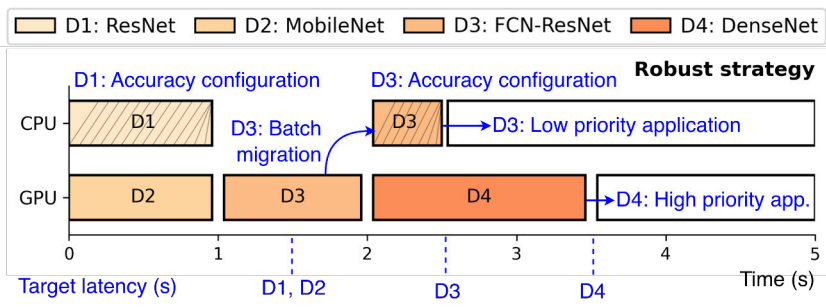

Figure 13. Run-time adaptive strategy jointly enforcing accuracy configuration, cluster mapping, and batch migration while considering application priorities

ordinated strategies [12; 63; 14] improve resource utilization by distributing tasks across CPU and GPU clusters depending on workload heterogeneity and by adapting DVFS. Nevertheless, these methods lack priority awareness, causing tasks such as *DenseNet* to be delayed, thereby increasing the latency of high-priority workloads. Consequently, in both approaches, many applications still fail to meet their latency objectives. This example highlights the need for a run-time resource management framework capable of context-aware tuning of available actuation knobs, thereby enabling requirements-aware scheduling decisions tailored to both workload characteristics and system dynamics.

3.3.3 Robust multi-DNN orchestration

A robust resource scheduling strategy actively tunes the available system and workload-actuation knobs to meet multi-DNN requirements. The same example from Section 3.3.2 is continued, where the inference requests of four different DNNs are introduced. A robust scheduling strategy is illustrated in Figure 13, which leverages accuracy configuration, batch migration, and DVFS while ensuring priority-aware workload inference to meet the required latency constraints. The robust strategy uses a lighter version of *D1* (ResNet-50) to meet the CPU cluster's latency constraint. Upon the arrival of a higher-priority *D4* application, the strategy migrates the lower-priority *D3* to the CPU, freeing GPU resources for *D4*. It then chooses a lighter variant of *D3* (FCN-ResNet50) to meet the CPU cluster's latency requirements and adjusts the frequency to enhance overall energy efficiency. This example highlights the need for a run-time resource management framework that can perform context-aware tuning of available actuation knobs, thereby enabling QoS-driven scheduling decisions that align with both workload characteristics and system dynamics.

3.4 Control knobs for Multi-DNN orchestration

Multi-DNN orchestration exposes a multidimensional control space that directly governs the energy, latency, and accuracy characteristics of the DNN applications. This section formalizes the principal actuation knobs that enable adaptive run-time coordination across heterogeneous resources while satisfying application-level QoS.

3.4.1 Cluster selection and DVFS

Mobile HMP systems offer a range of energy-performance trade-offs across their CPU and GPU clusters. Typically, GPU clusters provide superior performance for DNNs; however, CPUs can still deliver reasonable performance, especially for lightweight DNNs [71]. Moreover, employing DVFS allows fine-grained control over energy-performance trade-offs. Multi-DNN inference strategies thus necessitate run-time cluster selection and DVFS configuration to meet QoS requirements.

3.4.2 Accuracy configuration

For evaluating accuracy, we use task-specific metrics—accuracy for image classification, mean Intersection-over-Union (mIOU) for object detection, and standard IOU for semantic segmentation. An inference task can be executed using different DNN models, each characterized by distinct compute intensity and inference accuracy. Additionally, every DNN model may include multiple (quantized, pruned, or shallower) variants. This introduces a broader accuracy-performance trade-off space that can be leveraged by selecting an appropriate DNN model and its corresponding variant. Multi-DNN inference strategies must configure task inference accuracy at run-time by navigating this trade-off space while accounting for the diverse QoS requirements.

3.4.3 RL-based online exploration

Multi-DNN orchestration on heterogeneous edge platforms requires cluster selection, accuracy configuration, DVFS actuation, and batch migration, all under dynamic workload conditions. The stochastic nature of inference request arrivals, varying application priorities, and fluctuating resource availability make the scheduling problem NP-complete. A robust decision-making mechanism is required to navigate the extensive trade-off space of latency, energy, and accuracy objectives. RL provides a principled framework for generating run-time scheduling policies through continuous interaction with the environment. An RL-based agent can autonomously determine optimal resource configurations at run-time by observing system states and receiving reward feedback reflecting the workload's QoS compliance and energy ef-

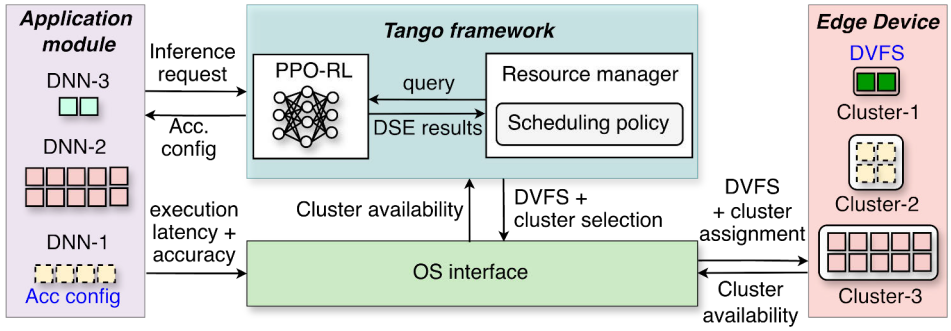

Figure 14. Tango framework showing the resource manager collecting requests and invoking PPO-RL for cluster selection, accuracy configuration and DVFS for DNN 1-3.

ficiency. Consequently, RL enables robust, context-aware scheduling decisions that generalize across diverse workload scenarios, thereby ensuring sustained QoS performance under dynamic operating conditions.

3.5 Evaluating Tango framework

This dissertation presents *Tango* as a robust multi-DNN orchestration framework for mobile and edge HMPs, managing dynamic workload fluctuations while satisfying the latency and model-accuracy constraints of inference requests.

Tango combines reinforcement learning with lightweight heuristics to jointly optimize performance and energy consumption. The framework exposes three primary actuation knobs, including cluster selection, model-level accuracy configuration, and cluster-level DVFS control. A PPO-based RL agent performs design-space exploration to determine the optimal cluster mapping and accuracy configuration for concurrently running applications, accounting for cluster availability and per-application latency and accuracy requirements. Following this RL-driven global decision, *Tango* applies a heuristic DVFS tuning phase to improve energy efficiency by adjusting the cluster frequency based on the gap between the current and target latency. Specifically, DVFS scales the operating frequency to minimize this latency deviation while avoiding unnecessary over-provisioning, thereby reducing energy consumption. The resource manager operates as a feedback control loop with Observe, Decide, and Calibrate phases, enabling adaptive and scalable scheduling of dynamic multi-DNN workloads under strict latency and energy objectives.

Tango evaluation. *Tango* was implemented on the Jetson TX2 heterogeneous edge platform [72], which integrates a Pascal GPU, quad-core ARM CPUs, and dual-core Denver CPUs. The framework was developed in Python using `CGroups` and CUDA/TensorRT libraries. Evaluation was conducted on widely used image classifi-

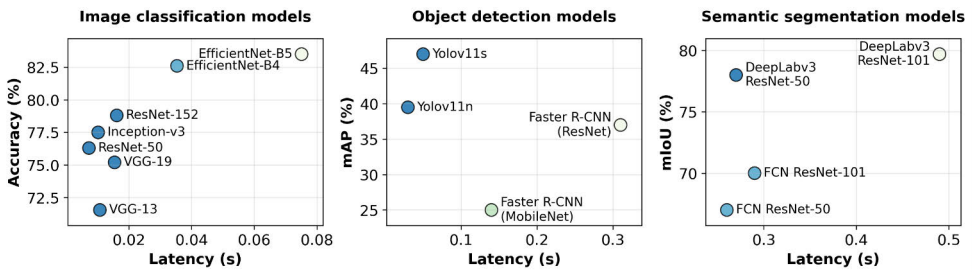

Figure 15. Profiling latency-accuracy trade-offs of different DNNs for each task

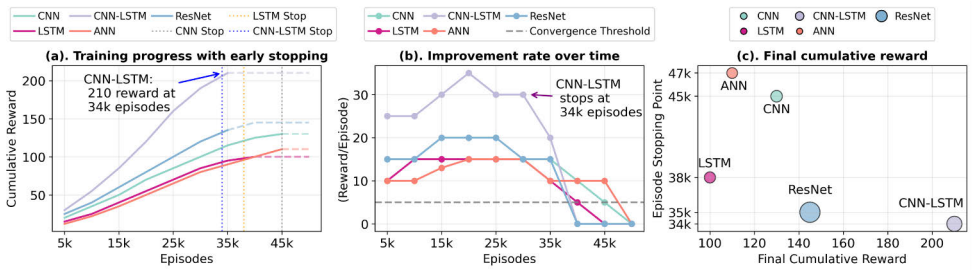

Figure 16. Comparative analysis of the performance of different RL agents

cation DNNs [73], including (i) VGG, (ii) MobileNetV2, (iii) ResNet, (iv) DenseNet, and (v) EfficientNet. The reinforcement learning controller employed a memoryless PPO agent without reward shaping. Performance was compared against default Linux governors, namely `Ondemand` and `SchedUtil`, as well as SoA multi-DNN schedulers, including Pipeit [22], OmniBoost [12], and Band [14].

The performance of *Tango* was evaluated under concurrent execution of two and three DNNs, reflecting realistic multi-application edge scenarios while respecting platform thermal limits. For two concurrent applications, experiments were conducted across multiple workload combinations with a 1s latency target, comparing against Linux governors and SoA multi-DNN schedulers. While baseline strategies relied on greedy placement, pipelining, or DVFS-centric optimization, they frequently failed to satisfy latency constraints and incurred higher energy consumption under contention. In contrast, *Tango* coordinated cluster placement, accuracy configuration, and DVFS tuning to consistently meet latency targets, achieving substantial reductions in latency and energy with negligible accuracy degradation. For three concurrent applications, where interference and resource contention significantly increased execution times and necessitated the use of lower-performance clusters, competing approaches exhibited pronounced latency violations. *Tango* maintained target compliance through adaptive accuracy scaling and resource orchestration. Overall, across all multi-DNN experiments, *Tango* improved latency and energy consumption by approximately 61% and 48%, respectively, at an average accuracy loss below 2%.

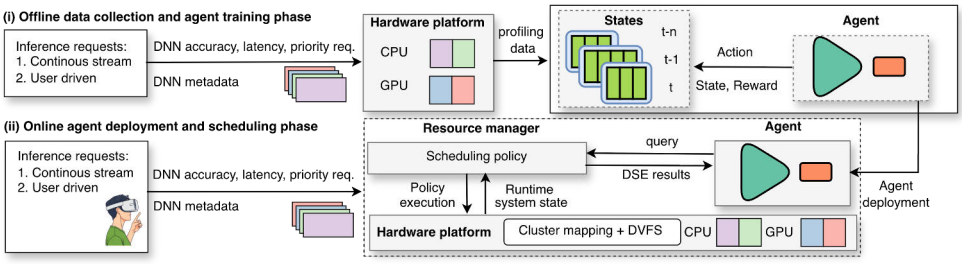

Figure 17. High-level workflow for *TangoX* framework: (i) offline CNN-LSTM agent training with profiled data, and (ii) online agent deployment and scheduling

3.6 Extending to TangoX framework

The capabilities of the *Tango* framework were extended to *TangoX* by accounting for application priorities and the platform's energy constraints for multi-DNN orchestration, through an advanced design-space exploration mechanism using Proximal Policy Optimization (PPO)-RL. The *TangoX* framework integrates offline profiling of DNN-cluster affinity with real-time monitoring of workload variations to jointly control cluster selection, model-accuracy adaptation, DVFS tuning, inter-cluster task migration, and inference prioritization during execution. Coordinating multi-DNN workloads across heterogeneous mobile/edge platforms is an NP-complete challenge due to the combined effects of resource heterogeneity, workload diversity (in compute demands, latency targets, and priority levels), and runtime variability. To efficiently explore this complex optimization space, an RL-based controller was implemented using the PPO algorithm to determine optimal resource configuration policies enforced by the *TangoX* scheduler. The proposed RL agent incorporates a hybrid architecture comprising a Convolution Neural Network (CNN) encoder and an Long Short-Term Memory (LSTM) decoder—where the encoder captures spatial dependencies across batched data, while the decoder models temporal dynamics arising from the unpredictable arrival and termination of inference tasks.

An overview of the *TangoX* framework is presented in Figure 17. The framework operates through two primary stages: (i) *Offline data collection and training*, and (ii) *Online agent deployment and scheduling*. During the training stage, two categories of inference requests were considered, including continuous streaming and user-driven. Each request specifies its QoS (accuracy and latency requirements), assigned priority, and application metadata. To gather training data, a diverse set of inference requests were executed on an HMP platform, measuring latency, accuracy, and energy consumption across both streaming and user-driven scenarios. This dataset mirrors real-world mobile inference patterns and serves as the foundation for building an RL environment where the agent learns optimal resource allocation policies by observing system states, receiving rewards, and refining adaptive scheduling

behaviors. Once training converges, the trained agent is deployed to a mobile or edge platform and integrated into the runtime scheduler, thereby forming a robust multi-DNN orchestration mechanism. This enables real-time management of incoming DNN workloads through intelligent cluster allocation, adaptive model selection, and dynamic frequency scaling, thereby accommodating fluctuating workloads and system constraints.

TangoX evaluation.

TangoX was evaluated on the NVIDIA Jetson Orin NX 8GB platform, featuring a hexa-core ARM Cortex-A78 CPU (up to 1.9 GHz) and an Ampere GPU with 1024 CUDA cores (up to 765 MHz), and supporting independent DVFS for both clusters under Ubuntu 20.04. The framework was implemented in Python using `CGroups` and CUDA/TensorRT. Benchmarking covered three major computer vision domains: image classification (ResNet, MobileNet, DenseNet), object detection (YOLOv11, Faster R CNN), and semantic segmentation (FCN, DeepLabv3), each characterized by QoS constraints on latency, accuracy, and task priority. The CNN LSTM-based RL agent was trained using stochastic mixes of two to five concurrent DNNs, combining user-driven and continuous streaming requests with varying SLOs to emulate realistic mobile inference workloads. Comparative evaluation included five SoA scheduling strategies: *PipeIt* [22], *Band* [14], *MoC* [63], *CarIN* [65], and *RankMap* [60].

The latency and accuracy of multiple DNNs were first profiled against the three image classification, object detection, and semantic segmentation tasks, as shown in Figure 15. A diverse model pool was used to produce a wide range of latency-accuracy trade-offs for each task, enabling the platform to select the best option at runtime. A comprehensive overview of training dynamics and architectural relationships across different RL agents is shown in Figure 16. The CNN-LSTM architecture achieves a superior cumulative reward of 210 while converging at 34k episodes, exhibiting exceptional sample efficiency of 6.77 reward units per thousand episodes compared to traditional architectures: CNN (2.89), LSTM (2.63), ANN (2.34), and ResNet (4.14). The hybrid approach exhibits the most aggressive initial learning rates (25-35 rewards per episode) compared with baseline agents' modest rates (10-20 rewards per episode) and converges to the convergence threshold the earliest. Despite having moderate model complexity (824k parameters), CNN-LSTM balances architectural sophistication with learning efficiency, outperforming both smaller models, such as an ANN (500k parameters), and larger alternatives, such as ResNet (1.5M parameters). The analysis indicates that intelligent architectural design, which combines spatial feature extraction via CNNs with temporal dependency modeling via LSTMs, is more critical than raw model size for achieving superior performance-to-training-cost ratios in resource-constrained deployment scenarios.

TangoX was evaluated against multiple SoA strategies against 2 and 3 concurrently running applications as shown in Figure 18. For the two-application work-

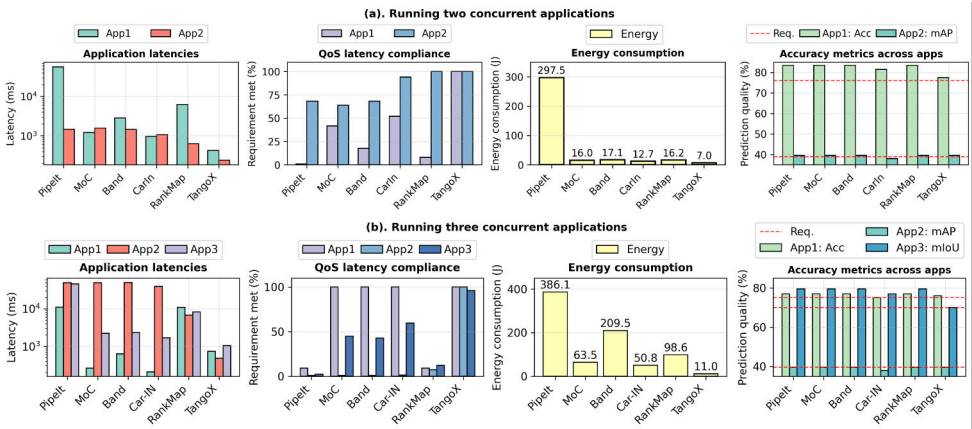

Figure 18. Comparative analysis of different strategies for 2 and 3 concurrent DNN inference workloads

load comprising image classification and object detection, TangoX achieves up to 99.3% and 83.6% lower latency for App1 and App2, respectively, while consuming 97.7% less energy than existing approaches. The framework successfully meets all QoS requirements by opportunistically migrating applications between CPU and GPU clusters based on priority, selecting appropriate model variants (InceptionNet-v3 for App1 and YOLOv11n for App2), and adjusting DVFS to optimize energy-performance trade-offs. In the three-application scenario involving image classification, object detection, and semantic segmentation, TangoX reports average latency reductions of 97.65%, 95.68%, 95.73%, 94.59%, and 91.34% compared to PipeIt, MoC, Band, CarIN, and RankMap, respectively. TangoX achieves an average 88.4% reduction in energy consumption by significantly reducing inference latency, demonstrating efficient power utilization while meeting stringent QoS constraints.

4 Compound AI Inference on HMPs

Contemporary AI deployment at production scale remains dominated by unimodal classification systems, in which models are designed to perform task-specific predictions on either the text or the vision modality [74]. Modern AI frameworks are rapidly shifting from unimodal inference to Compound Artificial Intelligence (cAI) architectures, which integrate task-oriented models and components to address complex problems [7; 75]. cAI systems combine inference requests of LLMs, encoder transformers, and DNNs to deliver new applications such as conversational language agents [75; 76; 74; 77], augmented and virtual reality (AR/VR) devices, and autonomous vehicles [78]. cAI systems provide compositional flexibility by dynamically connecting multiple transformer models (both encoder-based and generative) and DNNs during execution [79; 80]. There is a growing need to deploy cAI inference capabilities on mobile and edge platforms to overcome the latency, bandwidth, and privacy limitations of cloud-based infrastructure [71]. However, executing cAI workloads on heterogeneous edge devices presents significant challenges, particularly in scheduling concurrent transformer and DNN operations while minimizing inference latency within strict power budgets [14]. This chapter presents *Twill* as an online scheduling framework for deploying cAI inference workloads on heterogeneous multi-core platforms, managing run-time requests through heuristic exploration, and tuning configurable parameters to minimize end-to-end latency within a specified power budget. The previous chapters considered CPU-only platforms (Chapter 2) and CPU-GPU platforms (Chapter 3). Twill extends the platform's capabilities by adding a specialized Deep Learning Accelerator (DLA) [81] to CPU and GPU clusters. *Twill* framework was published in the following article:

IV **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Twill: Scheduling Compound AI Systems on Heterogeneous Mobile Edge Platforms*. International Conference on Computer-Aided Design (ICCAD), 2025.

The contributions include:

- *Twill*, a run-time framework designed for partitioning, distributing, and scheduling cAI workloads on heterogeneous mobile edge devices.
- An online heuristic model that profiles DNN, transformer, and LLM inference

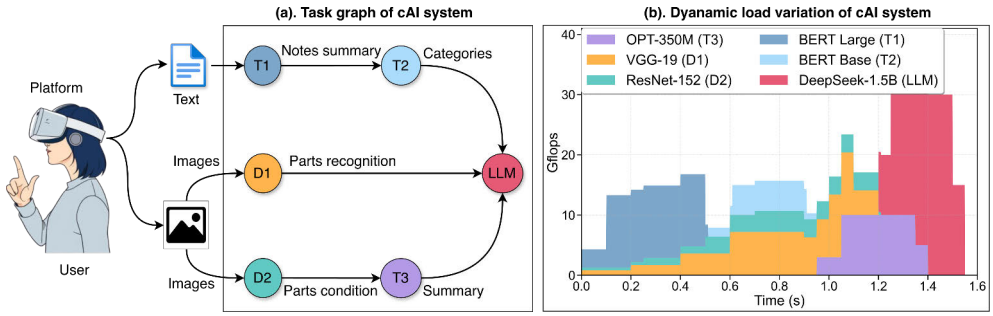

Figure 19. Exemplar cAI system. (a) Task graph for cAI system chaining multiple models, (b) Run-time workload variation and compute diversity

requests to identify the most suitable compute cluster (GPU/DLA) for each task.

- Mechanisms for mapping and migrating inference tasks across clusters, priority-aware task suspension to manage concurrent DNN, transformer, and LLM executions, and DVFS-based frequency control for power management.
- Performance evaluation of *Twill* on the Jetson Orin NX edge AI platform using modern cAI workloads.

4.1 Characteristics of compound AI workloads

Compound AI (cAI) enables complex multi-agentic, multi-modal intelligence by orchestrating multiple specialized models to collaboratively perform diverse tasks. Figure 19(a) illustrates a cAI workload (structured as a task graph) for generating a maintenance report from user-provided images and text inputs. In this setup, DNN models (`D1: VGG-19` and `D2: ResNet-152`) handle image classification and object detection, encoder-transformer models (`T1: Bert-base` and `T2: Bert-large`) perform text summarization and classification, and `LLMs: Deepseek-R1, OPT-350m`) are responsible for reasoning and report generation. Each model extracts key features from its input and passes them to subsequent components to collaboratively accomplish the overall task. Models `T1`, `D1`, and `D2` are independent inference tasks that can execute concurrently, whereas `T2`, `T3`, and the LLM depend on upstream outputs.

The exemplar cAI workload was deployed on the Nvidia Jetson Orin NX platform [82]. Figure 19(b) presents the performance requirements (in GFlops) of this system, showing that execution involves: (i) simultaneous operation of multiple DNNs and a transformer ($t = 0s$ - $0.9s$), (ii) concurrent execution of multiple DNNs and a generative transformer ($t = 0.9s$ - $1.2s$), and (iii) parallel inference of multiple generative transformers ($t = 1.2s$ - $1.5s$). This example highlights the computa-

tional diversity and dynamic workload variations of cAI systems. Consequently, an efficient workload scheduling strategy is required to infer cAI systems on resource-constrained edge platforms that can concurrently execute transformer and DNN models, with highly variable arrival times and diverse user requirements [79; 80].

4.2 Operational dynamics of NVIDIA's accelerators

Mobile edge platforms are resource-constrained due to limited energy budgets, but they support AI inference services through powerful GPUs and domain-specific DLAs [71; 83; 14]. The trending edge devices feature GPUs, NPUs, or Tensor Processing Units (TPUs) [84] in smartphones such as iPhone [4], smart glasses including Ray-ban Meta [85], and AR gear including Apple Vision Pro [5]. The NVIDIA Orin NX 8Gb platform was considered for the exploration study, which is equipped with an Ampere GPU and one NVDLA.

4.2.1 NVIDIA's GPU architecture

NVIDIA GPUs provide parallel computational architectures optimized for matrix MAC operations, which are fundamental to AI workloads. NVIDIA GPUs were initially designed for graphics and have since evolved to specialize in AI acceleration, featuring dedicated tensor processing units and CUDA cores that maximize computational throughput. Modern AI workloads have billions of parameters that require independent matrix calculations during both training and inference. The model parameters contribute to the matrix weights, which must be updated iteratively. NVIDIA tensor cores perform computations in parallel by executing distributed MAC operations as a single unit, without decomposing them into separate multiply operations.

The NVIDIA Ampere GPU architecture, as illustrated in Figure 20, spans edge-based solutions in the Orin series to cloud-grade variants. The Ampere series features multiple Streaming Multiprocessors (SMs) that execute parallel instances of the workload by exploiting the available CUDA cores. NVIDIA Ampere is a significantly higher-performing series than its previous Volta series. Ampere features 7 nm transistor processes with third-generation Tensor Cores that support higher-precision TF32 and BF16, as well as structured sparsity. The Ampere series enables distributed inference via Multi-Instance GPU (MIG) and Multi-Process Service (MPS), which partition a single GPU into multiple isolated instances. However, these strategies are not supported for the Orin series.

4.2.2 NVDLA architecture

The NVDLA is designed to accelerate DNN operations while simplifying system integration [81]. DNNs are predominantly composed of convolutions, activations,

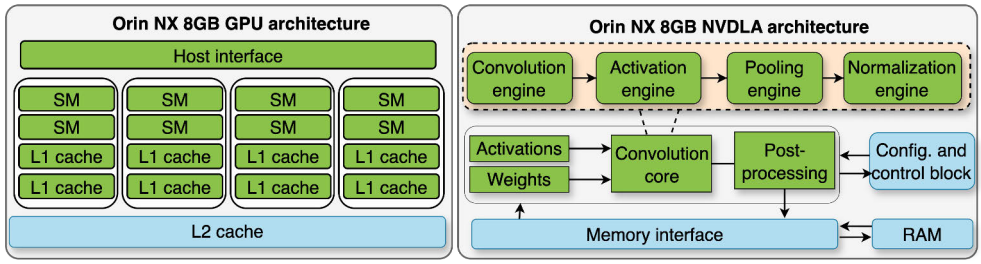

Figure 20. GPU and NVDLA architecture of Orin NX 8GB platform

pooling, and normalization operations, which exhibit predictable memory access patterns and fixed information flow, making them well-suited for specialized hardware acceleration. The hardware architecture, as shown in Figure 20 (b), comprises several independently configurable functional blocks. These include a convolution core optimized for high-performance convolution operations and a single data processor that provides single-point lookup capabilities for activation functions. A planar data processor handles planar averaging for pooling operations, while a channel data processor supports multi-channel averaging for advanced normalization functions. In addition, dedicated memory and data reshape engines accelerate tensor transformation and copy operations. Each block operates independently, with fine-grained scheduling boundaries, enabling system designers to scale individual components to meet specific performance requirements without modifying other units in the accelerator. NVDLA implementations are categorized into two types: head and headless. In the headless implementation, the system processor manages unit-by-unit hardware control, which is suitable for cost-sensitive, purpose-built devices that execute single tasks with an acceptable context-switch overhead. The head implementations incorporate a dedicated microcontroller tightly coupled to the NVDLA subsystem, delegating high-interrupt-frequency tasks away from the central processor to enable high-performance concurrent workloads in IoT devices.

The NVDLA hardware architecture supports two distinct operational modes, independent and fused, that fundamentally affect system performance and resource utilization. Independent mode treats each functional block as an isolated processing unit that performs memory-to-memory operations. Each block is configured individually for specific tasks, analogous to independent layers in deep learning frameworks, and requires data movement between the main system memory and dedicated SRAM during operations. Fused mode assembles multiple blocks into a processing pipeline, bypassing memory round-trip by communicating through small FIFOs, allowing the convolution core to pass data directly to the Single Data Point Processor, which forwards it to the Planar Data Processor, and subsequently to the Cross-channel Data Processor, significantly improving performance by eliminating memory access overhead.

4.2.3 Latency and energy characteristics of accelerators

The CPU clusters support sequential, single-threaded operations and have limited capacity to exploit dense MAC operations for a given DNN model. The GPU cluster is efficient for parallel operations, given the multiple cores provided by the dedicated SMs. The accelerators are designed to execute domain-specific operations. For example, NVDLA is efficient at performing convolution operations at low energy cost. Figure 21 shows the latency (s) and energy (J) for three different models: VGG-19, Yolo-v11s, and FCN-ResNet-101, which represent image classification, object detection, and semantic segmentation tasks, respectively. Latency and energy trends are typically similar across tasks, with the GPU cluster performing best. Therefore, the latency and energy characteristics determine the cluster affinity of a given DNN request.

4.3 Challenges of cAI inference on edge

cAI workloads exhibit highly variable computational requirements over time, driven by task-specific input modalities, dynamic dependencies among submodels, and user-defined latency constraints. Edge platforms, characterized by limited power and resource availability, must therefore sustain the concurrent execution of mixed workloads while minimizing latency. The heterogeneity of model architectures, input dimensions, and execution dependencies poses challenges for predicting run-time resource utilization. Hence, there is a critical need for adaptive run-time management that accounts for workload diversity and reconfigures execution strategies.

4.3.1 Model-workload affinity and operator-level support

Heterogeneous edge platforms typically integrate multiple compute clusters with distinct performance and operator-level support characteristics, resulting in varying degrees of affinity for different model types. NVDLAs are optimized for DNN workloads, supporting convolution and pooling operations, but lack support for transformer-specific components such as attention and normalization layers. Table 4 shows the

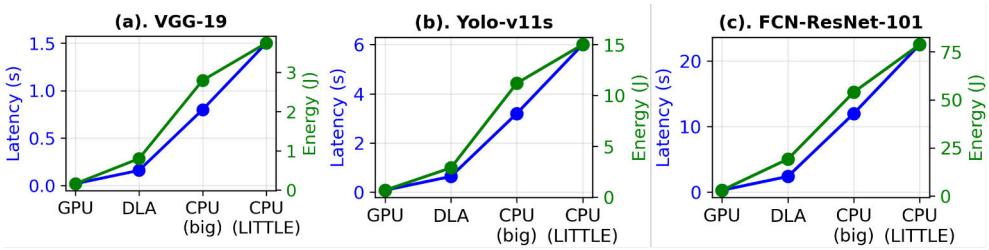

Figure 21. Cluster-wide latency and energy of (a). VGG-19, (b). Yolo-v11s, (c). FCN-ResNet-101

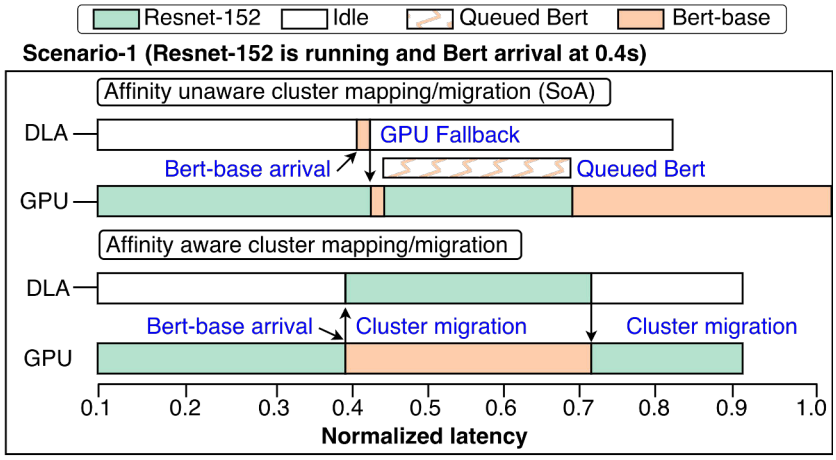

Figure 22. Various knob actuation scenarios, highlighting the cluster-affinity in SoA.

supported and unsupported operations in NVDLA. It should be noted that dynamic transformer operations are not typically supported by NVDLA.

Table 4. Supported and unsupported operations on DLA cluster.

Operation	Supported ✓	Unsupported / Limitations x
Convolution	INT8, FP16	2D only, kernel $\leq 32\times32$
Activations	ReLU, Sigmoid, Tanh	MatMul, Softmax, GELU
Pooling	Max, Avg	Window ≤ 8, stride ≤ 16
Normalization	Across-channel LRN	LayerNorm, BatchNorm
Reshape	Slice, split, merge, shuffle	4D tensors only

Therefore, determining the optimal cluster-workload affinity is crucial for minimizing inference latency and avoiding resource contention. Figure 22 shows different strategies that map DNN and transformers on GPU and NVDLA. A naive approach, unaware of model-cluster affinity, assigns the GPU to the first-arriving model for better performance and the free NVDLA cluster to the next-arriving model. In this scenario, if the second model is a transformer, it will fall back to the GPU cluster because NVDLA does not support those operations. Eventually, the transformer model will be queued and wait for the GPU cluster to become available. An intelligent, affinity-aware scheduling approach dynamically maps each workload to a cluster with the closest affinity and migrates the other applications to another cluster. Such affinity-driven scheduling enables concurrent execution of DNNs, transformers, and LLMs while meeting latency constraints across different scenarios.

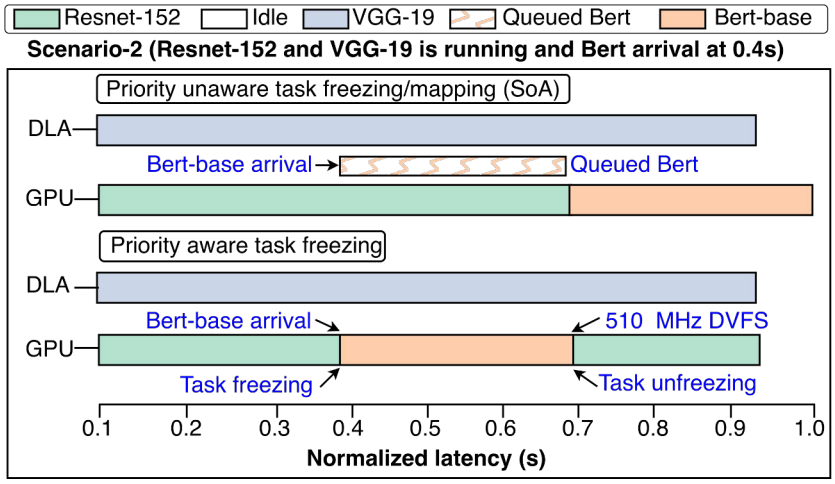

Figure 23. Various knob actuation scenarios, highlighting the priority awareness at run-time in SoA

4.3.2 Arbitrating application priority

In cAI systems, multiple inference requests can compete for higher-performing hardware resources. Depending on the functional importance and model type, inference requests may differ in their latency requirements. For instance, user-driven LLM prompts typically require immediate responses, whereas continuous DNN-based perception modules can tolerate brief interruptions. Therefore, establishing application-level priorities is crucial for efficient inference orchestration and an enhanced user experience. Figure 23 shows a working scenario where ResNet and VGG inference execution are occupying both GPU and NVDLA clusters, respectively. Priority unawareness will queue incoming transformer inference requests, risking high latency for higher-priority applications. Priority-aware scheduling allows the system to freeze lower-priority ResNets to make room for higher-priority Bert-base. Moreover, dynamic actuation of cluster-wide DVFS ensures overall high throughput for the workload. Such adaptive prioritization, supported by DVFS, is fundamental to ensuring low-latency responses for urgent tasks while maximizing overall system throughput.

4.4 Adapting existing strategies for cAI development

The existing edge AI orchestration techniques optimize the multi-DNN scheduling [83; 61; 12; 63; 22] and transformer models [24; 86] on mobile platforms with GPUs and DLAs. Some of these techniques do not consider transformer models and/or DLAs [61; 22; 63; 12; 15]. Using these techniques for cAI systems results in significantly higher latency and power consumption due to shared-resource contention

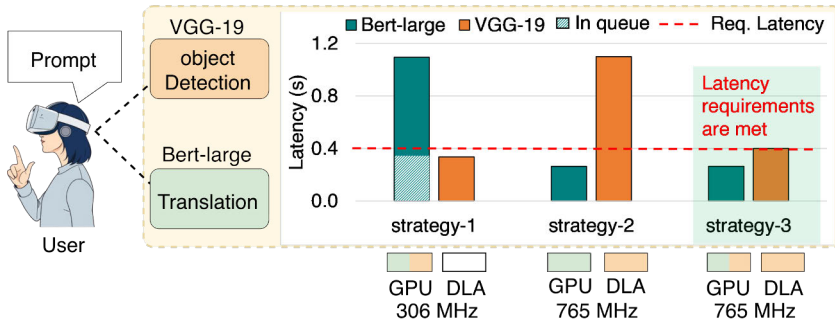

Figure 24. Inference latency of cAI system across different execution strategies

when mapping both DNN and transformer models to the GPU cluster. Techniques that leverage transformer models and DLAs [86; 14; 13] fail to address the lack of operation-level support for transformers on DLAs. Running transformers on the GPU is a reasonable choice when standalone inference of a transformer model is required. However, cAI workloads require concurrent execution of multiple DNNs and acp transformer.

Figure 24 shows different strategies for scheduling a first-arriving DNN model, VGG-19, and a later-arriving transformer model, Bert-base. The trivial strategy, based on the default Linux governors, maps both applications to the GPU, and the second-arriving application must wait until the GPU cluster becomes available for allocation. A more nuanced Strategy-2 maps the first application to the GPU and later-arriving applications to NVDLA. In this case, the application on NVDLA is relatively slow due to NVDLA 's low performance. Finally, strategy-3 maps the first application to the GPU, the second to NVDLA, and migrates the second application back to the GPU when it becomes available for allocation. This intelligent scheduling ensures that both applications meet their required latency targets. This dissertation presents a cAI orchestration strategy, *Twill*, that follows the intelligence of strategy-3.

4.5 Twill framework

Twill is an online scheduling framework to deploy cAI inference workloads on heterogeneous multi-core platforms. The platform handles run-time inference requests, performs online heuristic exploration, and adjusts configurable parameters to minimize overall latency while respecting a given power budget. As illustrated in Figure 25, the *Twill* framework consists of two main components: the *Model Interpreter* and the *Controller*. The *Model Interpreter* carries out online profiling to characterize each incoming inference request. It comprises a *Model Analyzer* to extract relevant information from model files and a *Compatibility Matrix* to verify whether model layers are compatible with the available clusters. For every inference request, the

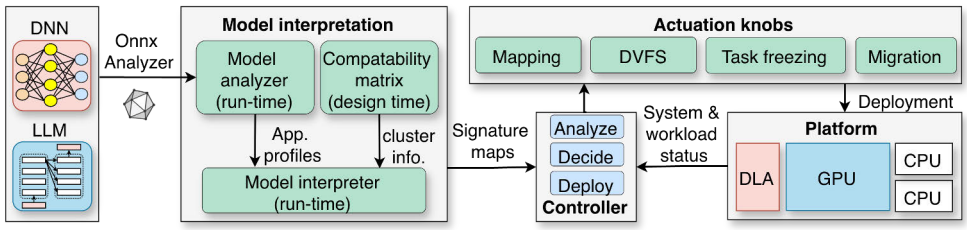

Figure 25. Twill Framework including Model Interpreter to analyze the model-cluster affinity and Controller module performing run-time control knobs actuation.

Model Interpreter generates a characterization of the executed model and produces a *Signature Map*, which is used to guide actuation decisions. The *Controller* observes the *Signature Maps* and overall system status to make decisions using predefined actuation knobs and a heuristic algorithm. It then applies the selected workload and system configuration and continuously monitors performance to ensure optimal operation.

4.5.1 ONNX-based model profiling

Twill first analyzes the model using the Open Neural Network Exchange (ONNX) structure and the cluster datasheet to generate layer-level signature maps for the given models.

Model analyzer. *Twill* employs *Model Analyzer* to parse the input ONNX file, profile the model structure using the *ONNX Run-time* library [87], and extract layer-level information and model metadata. The *Model Analyzer* also considers a user-defined *application priority*, which allows the *Controller* to identify suitable candidates for scheduling decisions under resource contention. This *application priority* is a user-configurable parameter set at design time. For instance, in typical cAI systems, user-driven prompt-based LLMs and transformer models are prioritized over DNNs processing streaming batches. *Twill* dynamically adjusts workload-scheduling decisions in response to changing application priorities. The *Model Analyzer* generates an application profiling table (`App_profile`) that combines extracted model information with application priorities. This `App_profile` table contains detailed model characteristics, including: 1) model name, 2) total parameters, 3) number of layers, 4) total floating-point operations, 5) layer types, 6) layer operation types, 7) layer-level input sizes, 8) layer-level output sizes, and 9) activation functions.

Compatibility matrix. *Twill* follows a DLA compatibility matrix (`dla_matrix`) derived from the official reference manual of Jetson Orin NX [81]. The matrix includes information on supported and unsupported operations, precision levels, layer types, spatial dimensions, batch sizes, kernel sizes, and padding and stride ranges. *Twill* requires cluster compatibility information because the available cluster may

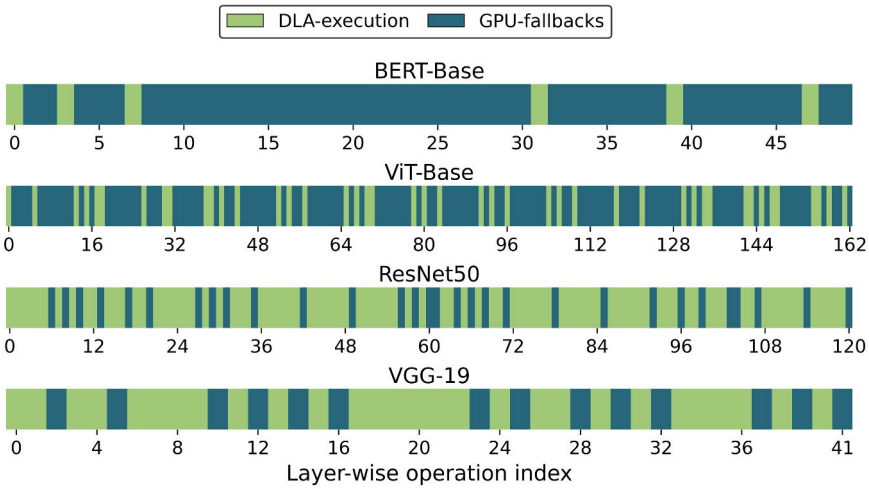

Figure 26. Supported operations on GPU and DLA clusters of the Orin NX platform

provide dedicated support for only a limited number of model operations. Figure 26 demonstrates the compatibility of inferring DNN and transformer models on the DLA cluster of Jetson Orin NX. The figure shows inference of two DNN models, including (`ResNet50` and `VGG-19`) and two transformer models (`Bert-base` and `ViT-base`) on the DLA cluster. Figure 26 presents layer-wise operators supported by the DLA and GPU fallback instances for unsupported operators. Most DNN inference operations of `VGG-19` and `ResNet-50` are supported on DLA. For transformer models `Bert-base` and `ViT-base`, DLA supports only the preprocessing and convolution operations, while most of the operations fall back to the GPU cluster. Therefore, the DLA cluster is better suited to running DNN workloads than the transformer cluster.

Model interpreter. The *Model Interpreter* determines the affinity between models and clusters using information extracted from the application profile and the compatibility matrix. Initially, the *Model Interpreter* retrieves layer details from the application signature table and sets up a compatibility map. By default, each layer is assigned GPU compatibility, since GPUs can execute all operation types. The `DLACompatible` function checks whether a layer can also run on the DLA by verifying three conditions: (i) the model's precision matches one of the DLA-supported precisions, (ii) the operation type is not listed among unsupported layers, and (iii) for convolution and fully-connected layers, additional constraints such as kernel size, stride, and padding fall within supported ranges. Finally, the *Model Interpreter* generates a *Signature Map* for each model, capturing the per-layer cluster affinity, layer type, number of floating-point operations, precision, and input and output shapes.

4.5.2 Scheduling policy of the controller

The scheduling policy is implemented in the *Controller*, which makes decisions based on the workload and system status. The *Controller* serves as a middleware layer that connects active workloads to the underlying hardware. The *Controller* interacts with OS interfaces to monitor system metrics, including power consumption and the operational state of different clusters. To manage workload deployment and execution, *Twill* operates advanced actuation knobs via OS interfaces, allowing the controller to override default OS scheduling policies and DVFS governors. These knobs include *Affinity-cluster mapping*, *Cluster migration*, *Task freezing*, and *DVFS* control. *Affinity-cluster mapping* assigns an inference request to a specific cluster, with a maximum of one model per cluster at any given time. *Cluster migration* enables the splitting of model layers and the reallocation of portions of the workload to another cluster. *Task freezing* temporarily suspends a running application to release resources for other tasks. Finally, DVFS allows the controller to adjust the operating frequency of a selected cluster.

4.6 Experimental evaluation and analysis

The *Twill* framework is implemented in Python as a user-space process on Linux, continuously monitoring workload execution and system status, including runtime power consumption via onboard sensors. For evaluation, *Twill* was deployed on NVIDIA Jetson Orin NX [82], which combines an Ampere GPU, an NVDLA, hexa-core ARM A78 CPUs, and 8 GB of RAM. The platform provides real-time power measurements for each cluster and supports CUDA for GPU operations, with runtime DVFS control and a default *Performance* scheduling governor. Experiments were conducted with a thermal design power of 10W, and the framework adds an average overhead of 15ms when fully utilizing a single CPU core.

For evaluation, cAI workloads spanning vision, text, and language tasks were considered. Vision workloads included DNNs such as VGG-19, ResNet-50, and EfficientNet-B4, as well as vision transformers (ViT-base and ViT-large) [88; 67; 89; 90]. Text classification was represented by encoder-based transformer models (Bert-base and Bert-large) [91], while large language model inference was performed using DeepSeek R1 (1.5B parameters) and Gemma 3 (1B parameters). The workloads were orchestrated using LangChain [79], and inference was implemented with PyTorch [20] and torchvision [73]. Hugging Face [92] was used for transformer models, and Ollama [93] for large language models.

For comparison, *Twill* was evaluated against three state-of-the-art edge AI inference strategies. MapFormer partitions multi-DNN workloads across GPU and DLA clusters at design time while scaling DVFS for power and throughput, assuming all requests arrive simultaneously. MapFormer was implemented using a transformer-

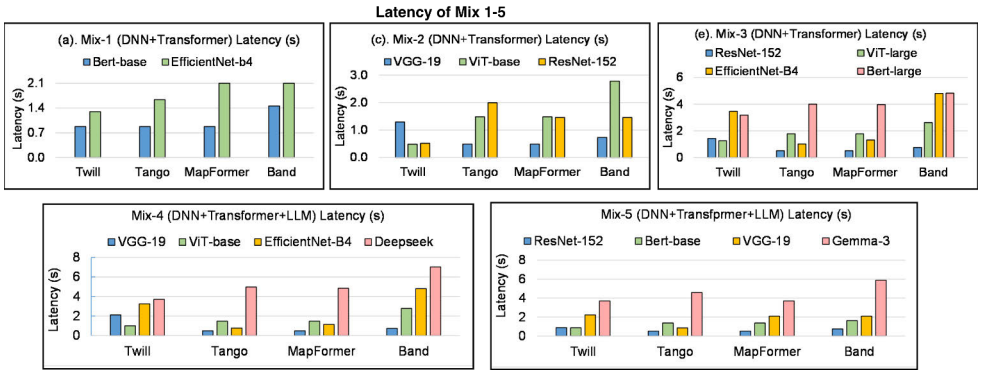

Figure 27. Latency of different strategies against different workload mixes.

Table 5. Workload mixes used for evaluation

Mix	Models
Mix-1	Bert-base, EfficientNet-b4
Mix-2	VGG-19, ViT-base, ResNet-152
Mix-3	ResNet-152, EfficientNet-b4, ViT-large, Bert-large
Mix-4	VGG-19, EfficientNet-b4, ViT-base, DeepSeek-R1
Mix-5	ResNet-152, Bert-base, VGG-19, Gemma-3

based estimator to predict throughput and power distributions. Tango operates at run-time on CPU and GPU clusters, utilizing reinforcement learning to manage multi-DNN workloads. The RL framework was implemented using the Gymnasium library [94]. Band partitions multi-DNN workloads into subgraphs based on supported operations and FLOP counts, mapping them to GPU and NPU clusters.

Performance was evaluated in terms of per-application inference latency, overall workload throughput, and platform power consumption. For DNNs, latency was measured per batch of input images; for transformers, per text prompt; and for LLMs, per generated output token. Throughput quantified the system's capacity to handle dynamic cAI workloads, capturing the effectiveness of *Twill* in optimizing inference across heterogeneous clusters.

4.6.1 Evaluation summary

To evaluate robustness under diverse composable AI workloads, *Twill* was benchmarked against SoA edge inference strategies across multiple workload mixes summarized in Table 5. These mixes incorporate varying combinations of DNNs, transformer models, and LLMs, introducing increasing levels of heterogeneity, concurrency, and dynamic arrivals to assess scheduling efficiency under realistic execution scenarios.

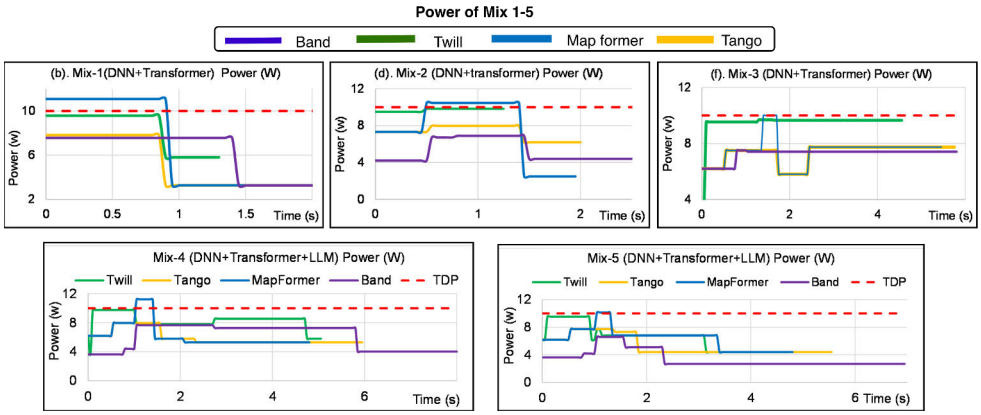

Figure 28. Power of different strategies against different workload mixes.

Based on the experimental results shown in Figures 27 and 28, Twill demonstrates significant performance improvements across all workload mixes compared to state-of-the-art edge AI inference strategies. Figure 27 illustrates the inference latency of different applications within each workload mix, while Figure 28 presents the run-time power consumption profiles throughout the execution. Twill consistently achieves the lowest execution time across all scenarios, with particularly notable improvements in Mix-2, where it reduces execution time by up to 54% compared to other approaches. Specifically, for Mixes 1-5, Twill achieves execution times that are 38%, 54%, 22%, 37%, and 31% lower than the best-performing state-of-the-art strategy, respectively. On average across all workload scenarios, Twill demonstrates 20%, 19%, and 34% lower execution times than Tango, MapFormer, and Band, respectively. The coordinated joint actuation of multiple control knobs for run-time varying cAI workloads is fundamental to Twill's superior performance. Unlike existing approaches that rely on offline analysis and design-time decisions, Twill continuously monitors workload characteristics and system status to make adaptive scheduling decisions. Based on the monitored information, *Twill* makes nuanced decisions on cluster selection, migration, and DVFS settings. Other strategies, such as Tango, MapFormer, and Band, are limited to offline analysis of workload characteristics and can only handle inference request arrivals known at design time. Hence, they may minimize the inference latency of the first- and/or second-arriving applications, but result in significantly higher overall inference latency as the number of concurrent inference requests and compute diversity increase. Twill's run-time adaptability becomes increasingly valuable in complex scenarios where multiple heterogeneous models with unknown arrival patterns compete for limited resources.

5 Distributed DNN Inference over heterogeneous edge clusters

DNNs require processing of continuous input data streams from multiple sources to enable IoT, smart video analytics, autonomous and assisted driving systems, wearable health monitoring, and augmented and mixed reality. The computationally intensive DNN tasks are commonly offloaded to powerful cloud servers for remote inference [71]. However, cloud-based inference requires persistent data transmission between the source device and the cloud server, degrading QoS due to increased transmission overheads, potential disconnections, and data privacy risks [11; 17]. As discussed in previous chapters, DNN tasks can be executed directly on source devices through efficient resource allocation. However, with limited computational capacity, low-latency inference is only possible on the source device for a limited number of input batches. Computationally intensive DNN workload can be distributed across a cluster of locally available edge devices for collaborative execution [10]. Distributed edge inference improves inference responsiveness by leveraging nearby trusted devices. An edge cluster comprises heterogeneous devices with diverse hardware architectures, compute capabilities, and energy-performance characteristics [95]. Device-level heterogeneity complicates workload distribution, as DNNs should be partitioned asymmetrically based on the computational capacity of available nodes [83]. Existing strategies partition the workload into equal-sized blocks and assign each block to a local node, without accounting for cluster-wide heterogeneity. Consequently, devices with lower computational capacity take longer to complete their assigned blocks, while devices with higher computational capacity finish earlier and remain idle until global synchronization is achieved. Uniformly sized partitions lead to performance imbalance and underutilization of cluster nodes. Beyond workload distribution, approximation can also enable a trade-off between inference accuracy and performance, providing additional leverage to meet performance constraints [96]. However, combining accuracy–latency trade-offs with non-uniform workload distributions across heterogeneous devices significantly increases the complexity of the design space. As a result, identifying effective workload-splitting points is challenging. Hence, determining an efficient workload distribution strategy requires reasoning jointly about device capabilities, partition sizes, and model accuracy configurations. The previous chapters presented nuanced run-time resource management strategies for AI orchestration on a single edge device. This

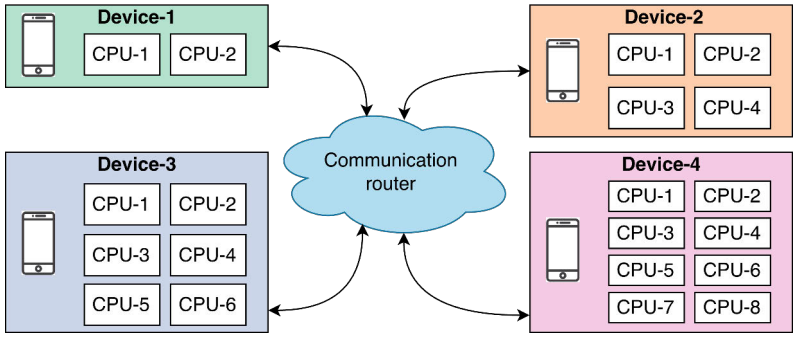

Figure 29. A heterogeneous edge cluster for collaborative execution of workloads

chapter presents a distributed edge framework for streaming applications that leverages approximation and data splitting to maximize inference throughput. The framework is presented in the following publication:

V **Zain Taufique**, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *Adaptive workload distribution for accuracy-aware DNN inference on collaborative edge platforms*. Asia and South Pacific Design Automation Conference, 2024.

The novel contributions include:

- Development of an edge cluster framework enabling real-time monitoring of system behavior and workload allocation,
- Design of a heterogeneity-aware, adaptive workload distribution strategy that balances accuracy and performance,
- Implementation of the proposed framework on actual edge devices (Odroid XU4, Raspberry Pi 4, and Jetson Nano) and its evaluation against leading workload distribution methods.

5.1 Significance and challenges of DNN workload distribution

A collaborative edge cluster is a network of nearby, trusted edge devices that share computational resources and exchange intermediate inference data [95]. Collaborative edge inference relies on the ability of distributed edge devices to share run-time resource availability, synchronize execution, and coordinate workload distribution across the network edge [24]. Figure 29 illustrates a conceptual setup of collaborative edge devices. The devices (Device-1 to Device-4) are interconnected via a communication router, such as a Wi-Fi router, forming a local edge cluster. The communication router facilitates data exchange and coordination among the devices

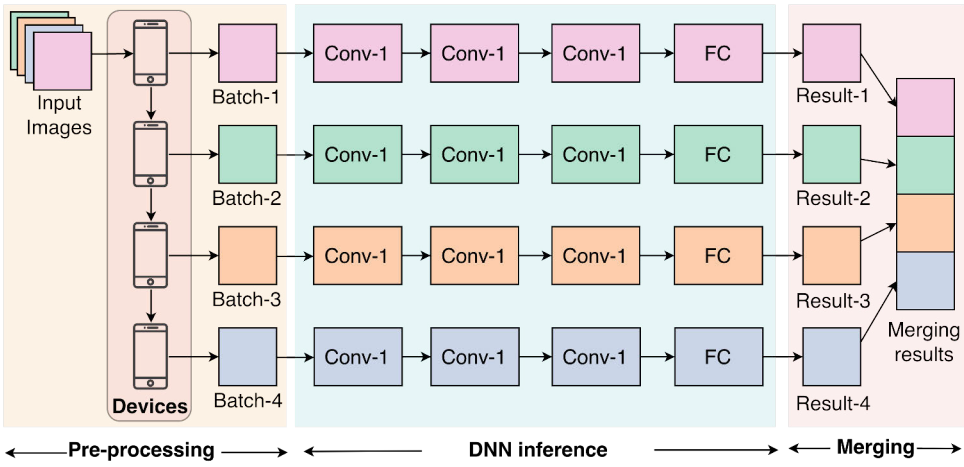

Figure 30. Splitting the input batches for distributed inference

to perform collaborative tasks. The devices have heterogeneous computational capacities with varying numbers of cores. Workload distribution across the cluster enhances throughput without compromising the privacy concerns associated with remote cloud offloading [22; 95]. Such collaboration enables dynamic distribution of inference tasks based on instantaneous compute capacity, rather than static device roles, allowing the system to adapt to fluctuating workloads and resource contention.

5.1.1 Orchestrating workload distribution

Collaborative edge inference requires adaptive workload-distribution strategies that account for system-wide workload dynamics, including available processing capacity, energy levels, and network conditions [16]. Input data arrives at the source device, requiring inference across multiple batches while meeting QoS requirements for latency and accuracy. The run-time workload orchestration strategy at the source device monitors the workload size, QoS requirements, and computational resource availability. The strategy determines whether execution should be performed entirely on the source device or distributed across the available edge cluster nodes. Input image batches are split and distributed to edge nodes for parallel execution, and each node reports its inference results back to the source device. Figure 30 demonstrates how each node receives the workload share, executes it and returns the results to the source device. Each device independently executes its assigned subset of the workload and produces intermediate inference results, which are later merged to form the final output. Consequently, workload distribution maximizes resource utilization and provides a scalable means to accelerate DNN inference on distributed edge platforms. This setup enables simultaneous computation across multiple devices, sub-

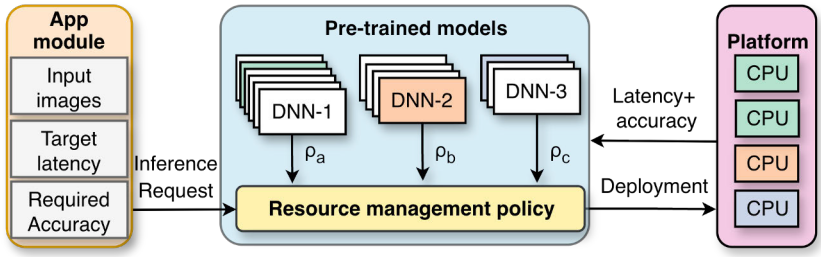

Figure 31. Accuracy configuration through model selection

stantially increasing overall throughput relative to sequential or single-device processing and ensuring synchronized output.

Exploiting accuracy-performance tradeoffs In practical edge deployments, inference requests often specify performance and accuracy requirements. These requirements may vary dynamically due to changes in workload intensity, input batch size, or application context. Fixing the accuracy configuration of a DNN model at design time limits the system's ability to adapt to run-time variations and leads to inefficient resource utilization. Enabling run-time accuracy configuration introduces an additional control that can be jointly optimized with workload distribution to satisfy application requirements under resource constraints.

Accuracy can be configured at the model level through the selection of pre-trained DNN kernels exhibiting distinct accuracy-performance characteristics. As shown in Figure 31, a pool of pre-trained models that differ in computational complexity and output accuracy can be utilized to select the model for a given inference request. Multiple quantized or pruned variants of a DNN model can be considered, with higher discrete approximation levels corresponding to lower accuracy and higher performance. This approach avoids expensive retraining or structural modifications at run-time, making it suitable for resource-constrained edge nodes. During run-time, the system selects an appropriate model from this pool based on the required accuracy and performance constraints of the incoming inference request.

Since inference is performed in parallel on distributed input data, the global output accuracy is computed by aggregating the local inference results. Each node reports its local accuracy statistics to the source device, which computes the overall accuracy as the weighted average of the individual partitions. This aggregation reflects the proportion of input data processed by each node and ensures that the reported accuracy accurately represents the system-level inference outcome.

Examplar scenario To illustrate the complexity of this exploration space, profiling experiments were conducted on a heterogeneous edge cluster comprising Odroid XU-4, Jetson Nano, and Raspberry Pi-4 boards. The performance and power consumption of these devices were evaluated while running MobileNetV2 image classi-

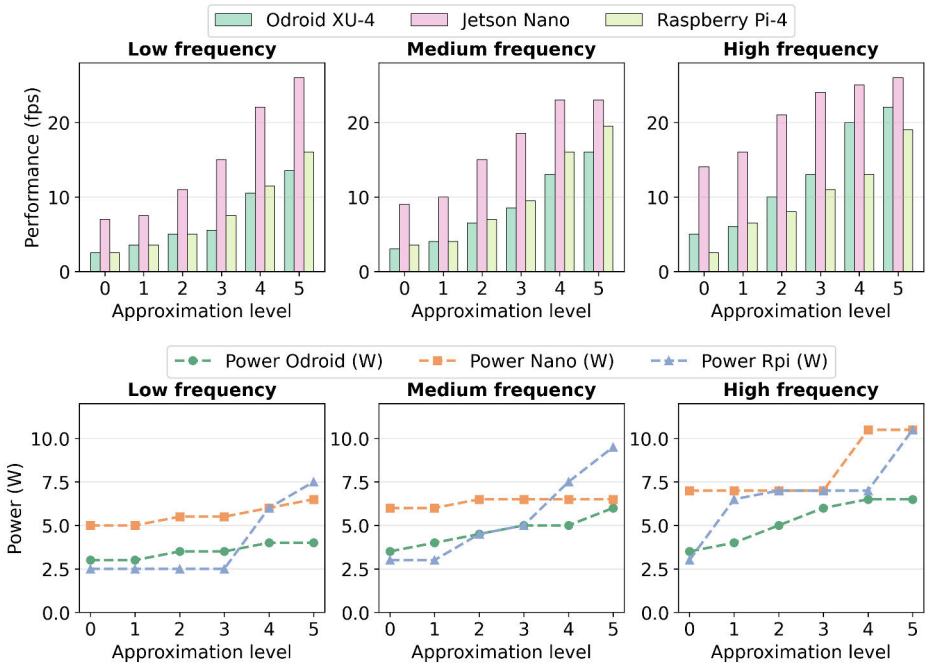

Figure 32. The power and performance comparison of distributing the MobileNetV2 inference workload among three edge devices

fication across six accuracy levels (approximation = 0–5, corresponding to progressively pruned models) under three DVFS frequency settings: low, medium, and high. Figure 32 (a) shows the inference throughput (frames per second) achieved by each device across different approximation levels and frequency settings. Jetson Nano consistently delivers the highest throughput across all configurations. All devices exhibit monotonically increasing performance as approximation levels increase, but the rate of improvement varies across devices, indicating device-specific sensitivity to model complexity. Frequency scaling has a pronounced effect on throughput, with high-frequency operation yielding 2-3× performance improvements relative to low-frequency settings across all devices. Figure 32 (b) reveals the corresponding power consumption characteristics. Jetson Nano exhibits the highest power draw, ranging from approximately 5W to 11W depending on frequency and approximation level. Odroid XU-4 exhibits lower power consumption, ranging from 3W to 6.5W, whereas the Raspberry Pi 4 consumes between 2.5W and 10.5W. Operating devices at various frequency settings (via DVFS) and selecting appropriate model configurations create a complex, multi-dimensional optimization space in which accuracy, throughput, and power must be balanced simultaneously across heterogeneous nodes. A workload distribution policy is required that distributes the workload proportionally to each device's peak capabilities. Jointly optimizing workload distribution, model

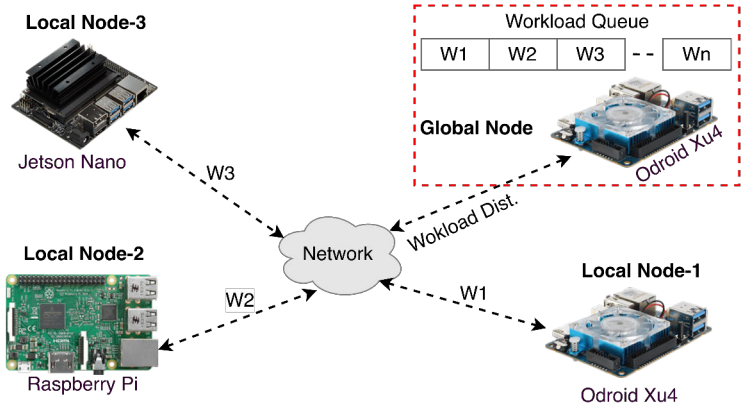

Figure 33. Workload distribution of inference requests among four edge devices

selection, and frequency scaling can achieve high performance while incurring minimal accuracy loss.

5.1.2 Coordination across edge cluster nodes

Effective scheduling and coordination across heterogeneous edge devices represents a critical challenge in distributed DNN inference systems. Edge clusters require sophisticated coordination mechanisms to orchestrate workload execution across multiple autonomous nodes with varying capabilities and availability. The distributed nature of edge computing introduces complexities in task allocation, execution monitoring, and result aggregation that must be carefully managed to ensure consistent performance and accuracy guarantees.

The edge cluster is hierarchically organized, with nodes that receive the inference request serving as leaders, distributing the workload to follower nodes. Figure 33 shows the leading nodes as Global Node (GN) and the follower nodes as Local Nodes (LNs). A heterogeneous edge cluster is considered to comprise four embedded boards: two Odroid XU4 boards, one Jetson Nano, and one Raspberry Pi 4. This hierarchical approach simplifies coordination by centralizing decision-making at the GN, which maintains a global view of cluster resources and workload requirements. However, this design also introduces potential bottlenecks and single points of failure that must be mitigated. The GN must efficiently handle incoming inference tasks by verifying the availability of the LNs, distributing heterogeneous workloads, and executing local inference, all while maintaining real-time responsiveness to dynamic system changes.

Edge cluster dynamics Edge devices may join or leave the cluster at any time due to mobility, battery depletion, thermal constraints, or network connectivity issues. Unlike static distributed systems, where resource availability can be assumed, edge

clusters must continuously monitor device availability and adapt workload distribution accordingly. When a device disconnects during execution, the system must rapidly reassign its workload to available nodes while maintaining performance and accuracy guarantees. This requires the GN to maintain up-to-date profiling information for all connected devices and to dynamically invoke the workload distribution policy in response to topology changes.

Temporal coordination presents another scheduling challenge. In data-parallel inference, where input batches are partitioned and distributed across devices, the target throughput is achieved by distributing the workload according to the nodes' computational heterogeneity. The proportional workload distribution strategy allocates smaller workload partitions to slower devices and larger ones to faster devices, aiming to equalize completion times across all nodes. However, achieving perfect load balance is complicated by variations in model complexity, input data characteristics, and run-time node availabilities. The system must therefore incorporate mechanisms to monitor execution progress and, if significant imbalances are detected at run-time, reallocate workloads.

Communication overheads and synchronization Communication overheads pose a significant challenge in distributed DNN inference, as the cost of data transfer across the edge cluster can exceed the target throughput of a given workload. In edge computing environments, where devices are typically connected via wireless networks with limited bandwidth and variable latency, communication latency becomes even more critical. The scheduling framework should carefully manage data movement between the source GN and the follower LNs to ensure a balanced computational-to-communication overhead ratio.

The primary source of communication overhead in data-parallel inference is the distribution of input data from the GN to the LNs and the subsequent synchronization of inference results. A workload partition should only be offloaded to a nearby edge device if the communication overhead of sending the data to a follower node is less than the computational overhead of executing the partition on the source node. Beyond the transfer of input data, the framework requires bidirectional communication among nodes for coordination and control. The GN must transmit workload assignments to each LN, and subsequently, the LNs must return inference results, including predicted labels, accuracy metrics, and performance statistics, to the GN.

For inference requests with small batch sizes, the time required to partition, transmit, and aggregate data may exceed the time saved through parallel execution, making distributed inference less efficient than local execution on a single powerful device. Conversely, for large-batch inference workloads typical of video analytics applications, the computational benefits of distribution generally outweigh communication costs, justifying the overhead. This workload-aware decision-making ensures that the system achieves the required performance across diverse operational scenarios while avoiding unnecessary communication when distribution provides marginal

or negative benefits.

5.2 Existing solutions for distributed inference

Existing methods for distributing DNN inference workloads across edge clusters employ either input data or DNN model splitting, depending on the specific application context [97; 11; 10; 17]. Traditionally, DNN workloads in an edge cluster are distributed using one of the following approaches: (i) *Uniform distribution* — dividing the workload equally among all devices [11; 10], (ii) *Asymmetric distribution* — allocating workloads in proportion to the computational capabilities of each device [97; 16; 17], or (iii) *Uniform distribution with approximation* — evenly distributing the workload while applying aggressive approximation to meet performance requirements [98]. Uniform workload distribution in heterogeneous environments leads to inefficient resource utilization, often degrading performance under heavy workloads. An asymmetric distribution that accounts for device capabilities yields better performance by optimizing resource usage; however, these strategies lack control over accuracy configuration. Conversely, uniform distribution with approximation can achieve higher performance for compute-intensive workloads, but at the cost of significant accuracy loss, potentially violating application accuracy requirements. These methods typically address either (i) accuracy performance trade-offs under the assumption of homogeneous edge nodes, or (ii) heterogeneous edge nodes while neglecting accuracy performance trade-offs. As a result, their effectiveness is limited under dynamic conditions, including run-time workload fluctuations and variations in device availability. To overcome these limitations, a *Proportional* workload distribution strategy is needed that achieves near-optimal workload distribution while satisfying performance and accuracy constraints through disciplined tuning of accuracy. The *Proportional* strategy must jointly optimize workload distribution and accuracy configuration to satisfy performance and accuracy requirements, while accounting for the heterogeneity and availability of edge nodes.

The workload distribution of a compute-intensive DNN inference is shown using the *uniform*, *uniform+apx*, *asymmetric*, and *proportional* approaches on an edge cluster comprising Raspberry Pi 4, Jetson Nano, and Odroid XU4 boards, as illustrated in Figure 34 (a). For each approach, the stacked percentile of the distributed workload and the selected DNN model across devices is depicted. The block size within each stack indicates the partition size, the color represents the device, and the model index (a0–a5) denotes the applied approximation level. The resulting accuracy and performance for all strategies are presented in Figure 34 (b). The *Uniform* and *Uniform+apx* strategies allocate an equal amount of workload to all devices. While the *Uniform* approach maintains accuracy, it fails to meet performance goals. In contrast, the *Uniform+apx* approach meets performance targets at the expense of accuracy through aggressive approximation. In contrast, both *Asymmetric* and *Pro-*

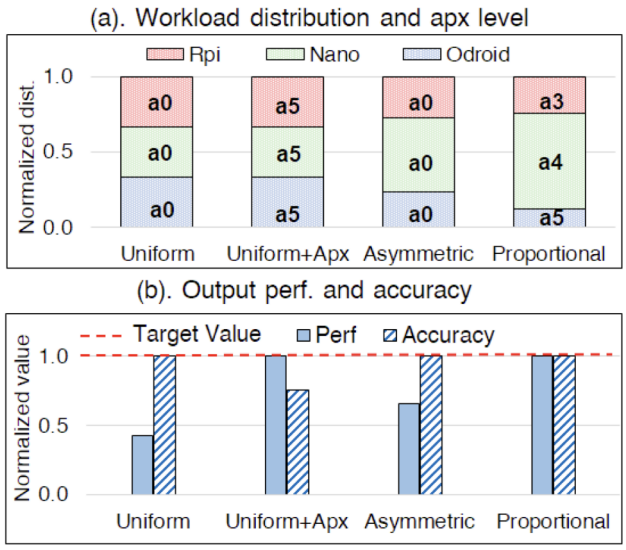

Figure 34. The workload distribution and accuracy configuration of MobileNetV2 workload among four different strategies

portional approaches assign an unequal amount of workloads. However, the *Asymmetric* method still fails to meet the performance requirements because the workloads exceed the device's capacity limits. The *Proportional* strategy, being heterogeneity-aware, successfully satisfies both accuracy and performance objectives by intelligently distributing and approximating the workload in proportion to each device's computational capacity. Figure 34 shows that (i) workload distribution and accuracy configuration must be jointly optimized to meet both performance and accuracy goals, and (ii) awareness of edge node heterogeneity is essential for effective coordination of workload distribution and approximation. The objective is to design an adaptive workload-distribution policy that implements the *Proportional* strategy to ensure optimal utilization of edge-cluster resources while meeting the requirements of DNN applications.

5.3 Proposed framework for distributed inference

Figure 35 illustrates the proposed adaptive workload distribution framework for collaborative heterogeneous edge clusters. The framework adopts a hierarchical structure composed of a GN and multiple LNs. The GN acts as a gateway and central controller responsible for global coordination, workload distribution, and inference dispatching, while the LNs executes local inference workloads according to the distributed configuration.

Accuracy configuration. Approximation is achieved by run-time model selection

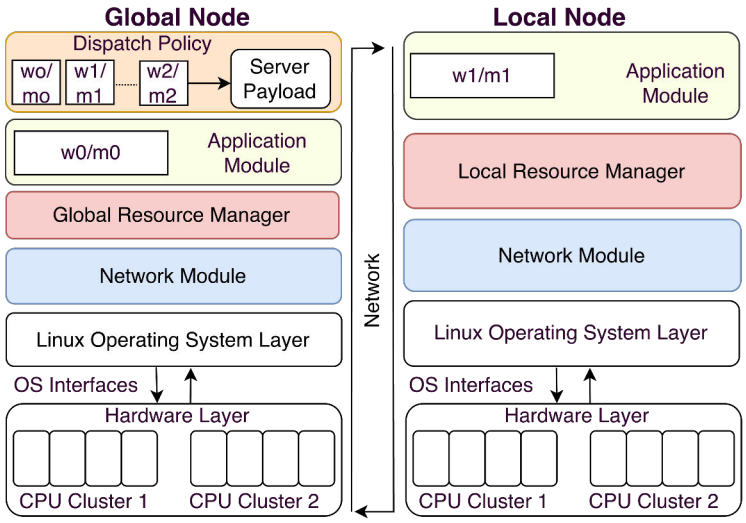

Figure 35. The proposed framework showing the internal architecture of both Global and Local nodes

from the pre-trained model pool. To enable run-time configuration of accuracy, the system performs an initial profiling phase on each edge node. During profiling, every available model variant is executed on representative input data to measure its achievable inference performance and output accuracy on the target hardware. The profiling results are recorded in a lookup table that captures the relationship between approximation level, device identity, performance, and accuracy. This profiling table serves as a compact representation of the exploration space for accuracy-aware workload distribution. Since the performance and accuracy of a given model vary across heterogeneous devices, node-specific profiling is essential to capture each model variant's true behavior. The profiling phase is performed once during initialization and updated dynamically if devices join or leave the edge cluster. At run-time, each inference request arrives with explicit performance and accuracy requirements.

The Global Resource Manager uses the profiling table to determine the minimum approximation level at which the cluster can collectively satisfy the requested performance while preserving the required accuracy. Accuracy selection is therefore not performed in isolation but is tightly coupled with workload distribution across the available nodes. For a given workload, the Dispatch Policy evaluates candidate configurations by examining each device's performance profile across different approximation levels. The policy identifies the lowest approximation level that ensures the aggregated cluster performance meets the target throughput. This ensures that the approximation is applied conservatively, only when necessary, to satisfy performance constraints. Once the approximation level is selected, the corresponding model variant is assigned to each workload partition and dispatched to the respective

edge nodes.

Resource management framework. At the GN, the *Dispatch Policy* jointly determines the workload partition and the associated approximation level for each LN. The GN maintains a queue of inference requests, each annotated with required performance and accuracy parameters. Upon receiving a new request, the GN invokes the *Global Resource Manager*, which leverages the stored profiling table to evaluate available resources and select an appropriate distribution and approximation configuration. The chosen configuration is then communicated to the LNs through the *Network Module* and executed via the *Application Module*. The GN also supports real-time monitoring and reconfiguration in response to dynamic workload variations or device unavailability.

Each LN comprises a local software stack that mirrors the GN structure, including the *Local Resource Manager*, *Network Module*, and *Application Module*. The Local Resource Manager is responsible for profiling, executing the assigned workload, and reporting the achieved accuracy and performance metrics back to the GN. All nodes run Linux, enabling kernel-level interactions for monitoring system utilization, power management, and workload scheduling via OS interfaces.

The proposed architecture enables heterogeneity-aware, adaptive workload distribution by jointly optimizing workload splitting and model approximation. This design ensures that distributed inference tasks meet global performance and accuracy constraints, even under dynamic run-time conditions such as fluctuating workloads or node failures. The integration of the Global and Local Resource Managers, along with the Dispatch Policy, forms the core of the adaptive middleware that supports collaborative DNN inference across heterogeneous edge clusters.

5.4 Experimental setup and results

The framework was implemented on a four-node heterogeneous edge cluster consisting of two Odroid XU4 boards, a Jetson Nano, and a Raspberry Pi 4, all running Ubuntu 20.04. The framework, written in C++, operates as middleware, employing a POSIX-based client–server architecture with multithreaded communication over Ethernet. According to the technical datasheets for the Odroid XU4, Raspberry Pi 4, and Jetson Nano, the TDP limits were set to 8W, 9W, and 10W, respectively. The system runs image classification workloads using TensorFlow Lite and OpenCV Lite with pre-trained MobileNetV2 models. Model accuracy is dynamically configured at run-time via a pre-trained model selection mechanism that offers various trade-offs between accuracy and complexity.

The framework was compared against SoA workload distribution methods: the *Uniform* strategy [97] for evenly distributed workloads in homogeneous clusters, the *Asymmetric* strategy [17] for non-uniform distribution in heterogeneous clusters, and the *Uniform + Apx* approach [98], which combines uniform distribution with work-

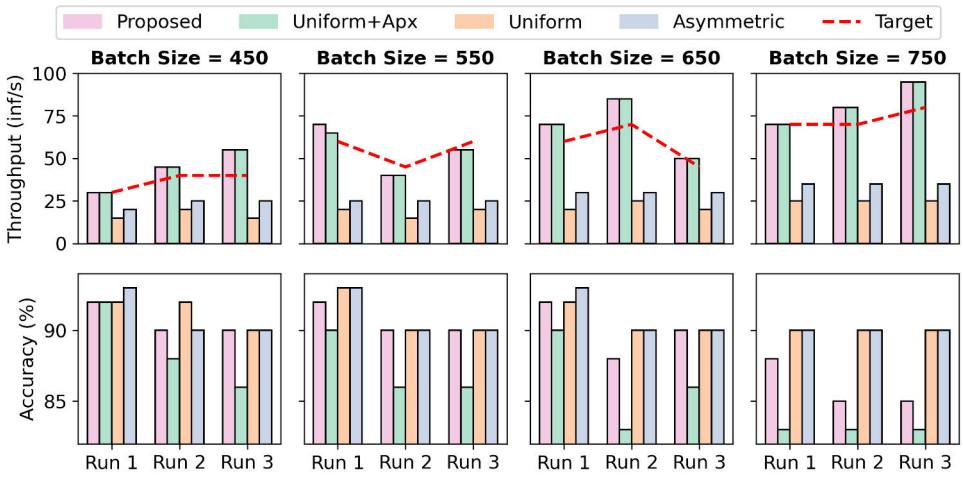

Figure 36. Experimental evaluation of MobileNetV2 inference across different workload sizes, throughput targets and accuracy requirements

load approximation. Each device was profiled to measure inference throughput and construct a performance table. All compared strategies were enhanced with dynamic voltage and frequency scaling (DVFS) to ensure compliance with TDP constraints. The workload-distribution mechanisms of these methods were isolated and applied to input-data splitting in the implementation. In addition, each strategy was adapted to accommodate dynamic workload variations to ensure a fair comparison.

Each strategy was evaluated under dynamic workload conditions, with varying performance and accuracy requirements and different input batch sizes. The resulting performance and accuracy for all experiments are illustrated in Figure 36 (a) and (b), respectively. The experiments use four distinct input batch sizes and three sets of performance and accuracy requirements for each batch size. The findings indicate that the *Uniform + Apx* strategy consistently meets performance goals but fails to maintain the minimum accuracy threshold due to its aggressive workload approximation. In contrast, the *Asymmetric* and *Uniform* strategies deliver higher accuracy but fall short of performance targets, as the workload demands exceed the edge cluster's rated capacity. In summary, the proposed strategy achieves near-optimal distribution and approximation, ensuring that performance constraints are satisfied while maximizing output accuracy. On average, the proposed approach yields a 52.63% performance improvement and incurs a 3.9% lower accuracy loss than other approximation strategies.

6 Distributed DNN inference with Hierarchical Partitioning

Distributed DNN inference on heterogeneous edge devices is commonly partitioned model- or data-wise and executed across multiple edge nodes to reduce end-to-end latency. Inference latency depends not only on inter-node workload distribution but also on the effective utilization of heterogeneous processors within each node during inference. Variations in model architecture, layer composition, and input characteristics further influence the suitability of different partitioning and scheduling choices across heterogeneous platforms. The existing approaches primarily focus on high-level workload distribution without accounting for the intrinsic core-level heterogeneity of edge nodes, which include diverse processing units such as CPUs, GPUs, and NPUs. Additionally, existing strategies do not jointly determine model and data partitioning decisions in accordance with the DNN architecture, system heterogeneity, and dynamic workload characteristics at both global and local levels. Consequently, current methods often lead to sub-optimal utilization of available resources, resulting in increased inference latency and inefficient workload scheduling across heterogeneous edge platforms. The previous chapter discussed how the distribution of input data and the configuration of accuracy can enhance overall inference throughput in streaming DNN applications. This chapter presents *HiDP*, a hierarchical partitioning framework that employs both data and model partitioning to enable distributed DNN inference across heterogeneous edge clusters. Unlike existing approaches, *HiDP* considers workload distribution at both the edge-cluster and node levels. *HiDP* framework was published in the following article:

VI **Zain Taufique**, Aman Vyas, Antonio Miele, Pasi Liljeberg, Anil Kanduri. *HiDP: Hierarchical DNN Partitioning for Distributed Inference on Heterogeneous Edge Platforms*. IEEE Design, Automation & Test in Europe Conference (DATE), 2025.

The contributions include:

- A hierarchical DNN partitioning approach designed to optimize inference throughput in distributed edge environments,
- A collaborative edge cluster framework enabling efficient local and global partitioning of DNN inference tasks,

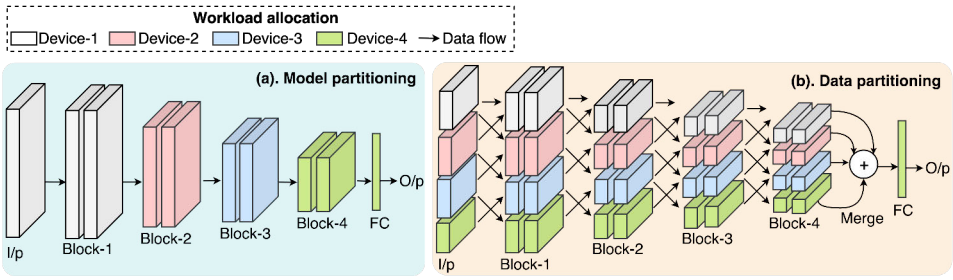

Figure 37. Data and model partitioning of an inference workload across four interconnected devices

- Comprehensive evaluation of the proposed HiDP approach against existing distributed inference methods on a real hardware edge cluster comprising Jetson Orin NX, Jetson Nano, Jetson TX2, Raspberry Pi 4B, and Raspberry Pi 5 devices.

6.1 Types of DNN partitioning strategies

DNN partitioning enables distributed inference across multiple heterogeneous edge nodes by dividing the DNN workload into smaller, executable blocks. Partitioning enables collaborative use of available compute resources, reducing inference latency and balancing workloads across edge clusters. As shown in Figure 37, DNN partitioning can be categorized into two major types, including *model partitioning* and *data partitioning*. Model partitioning divides the network into sequential blocks of layers, while data partitioning splits the input data into multiple smaller subsets. Both strategies aim to improve throughput and minimize latency, but differ in their granularity, overhead, and suitability for different workloads. Model partitioning is inherently temporal and operates in a pipelined manner, whereas data partitioning exploits spatial parallelism by distributing parallel sub-models across multiple processors. Selecting an appropriate partitioning strategy requires configuring partition points, block counts, and workload split ratios. Moreover, intelligent DNN partitioning requires careful consideration of the computation-to-communication ratio, available processing units, and the heterogeneity of the target hardware. The overall goal of partitioning is to optimize resource utilization, minimize end-to-end latency in distributed inference, and maintain model accuracy while reducing communication overhead.

6.1.1 Model partitioning with layer splitting

Model partitioning divides the DNN model into sequential blocks of layers, which are distributed across different devices for pipelined execution. As shown in Fig-

ure 37(a), the DNN layers are grouped into multiple blocks, where each block represents a contiguous set of layers. The layer blocks are allocated to different computational clusters or edge devices, which process a specific block and pass the intermediate activations to the next device in the sequence. This approach enables temporal parallelism across pipeline stages, allowing for continuous data flow and overlapping computation with minimal communication.

Model partitioning is particularly effective for dense DNN architectures with substantial inter-layer dependencies and deep hierarchies, where distributing consecutive layer blocks can balance computation across devices. It enables the dynamic creation of variable-sized DNN blocks based on the available compute resources within the cluster. The size of each block and the selection of partition points are key factors affecting latency and throughput. Blocks that are too small can incur excessive communication overhead due to frequent data transfers, whereas overly large blocks can cause compute imbalance and underutilization of available resources. Therefore, the optimal partition configuration must account for both the computational intensity of each layer and the inter-node communication characteristics.

6.1.2 Data partitioning with weights splitting

DNN applications operating on streaming inputs generate inference requests in the form of input batches that arrive at the source device for execution. These input batches can be split into smaller partitions, enabling parallel execution across multiple edge devices. In data partitioning, as shown in Figure 37 (b), the input data of a DNN inference request is divided into multiple smaller subsets that can be processed concurrently by replicas of the same DNN model across numerous devices. This approach enables distributed parallel inference by leveraging the available compute capacity of heterogeneous edge nodes. The fundamental principle of data partitioning is its ability to exploit spatial parallelism, in which each device executes the same model on a different portion of the input data and subsequently merges the partial results to produce the final output. After the input data is split, the resulting partitions are executed in parallel across multiple devices, and their outputs are merged through intermediate data aggregation operations to maintain overall inference accuracy.

Data partitioning is well-suited to workloads with large input sizes, such as image or video analytics. It enables the simultaneous use of multiple compute nodes, thereby reducing inference latency and improving overall throughput. However, this approach also introduces additional communication and synchronization overhead due to intermediate data exchanges among devices during the merging stage. The cost of merging results grows with the number of partitions and the size of intermediate feature maps, potentially offsetting the gains from parallelism if not carefully managed. Moreover, the feasibility of data partitioning depends on the DNN structure and input dimensionality—models with smaller input sizes or deeper ar-

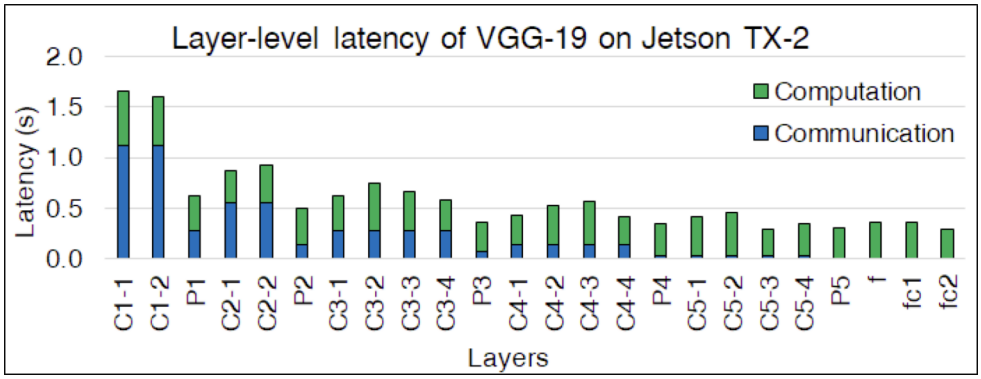

Figure 38. Layer-level latency of executing different layers of VGG-19 on Jetson TX2

chitectures may experience limited benefits due to communication-dominated execution times. Therefore, determining the optimal number of data partitions and the corresponding workload allocation among devices is crucial for balancing latency reduction and communication efficiency.

6.2 Global and local workload distribution

In the context of distributed DNN inference, *global workload partitioning* refers to decisions that determine how the inference workload is split and allocated across different edge nodes in the cluster. *Local workload partitioning* refers to how the workload assigned to an individual edge node is further partitioned and scheduled across its heterogeneous processing resources. This section discusses the challenges and significance of coordinating global and local workload distribution for efficient distributed inference.

6.2.1 Global workload distribution

The efficiency of global partitioning is determined by its ability to balance computation and communication overheads across the available edge nodes in the cluster. As discussed in chapter 5, global partitioning decisions must jointly consider workload splitting and assignment to minimize end-to-end inference latency. This requires reasoning about the trade-off between partition size and inter-node communication overhead. Finer partitioning enables more flexible workload distribution across heterogeneous nodes but increases the volume of intermediate data transfers, which can dominate execution time. Conversely, coarser partitioning reduces communication overhead but may lead to computational imbalance across the cluster, thereby limiting the effective utilization of available edge resources.

Figure 38 shows the layer-level latency of VGG-19 on Jetson TX2. The green

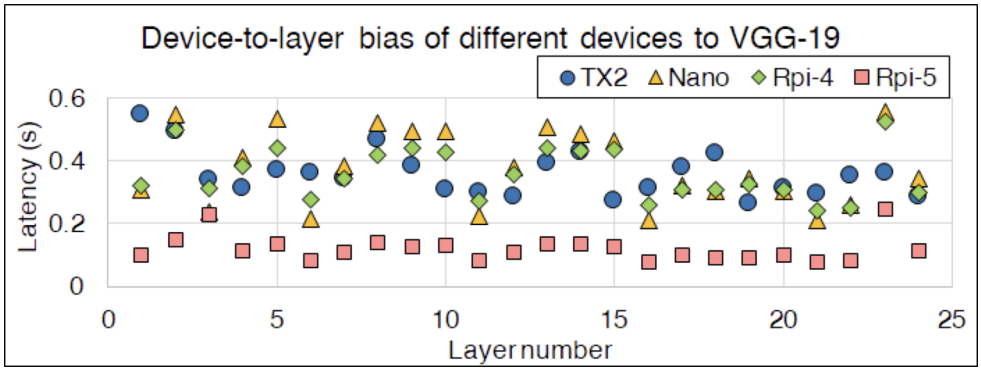

Figure 39. Layer to device bias of a model across multiple heterogeneous devices

bars represent the computational latency of executing the layer on the Jetson TX2. In contrast, the blue bars indicate the communication overhead of offloading the layer to an available cluster node. The plot shows that the initial convolutional layers are better suited for execution on the source device, while later layers can be offloaded due to low communication overhead. An intelligent scheduling strategy performs Design Space Exploration (DSE) over all possible partitioning points and determines the global workload offloading to minimize communication overhead.

An essential aspect of global partitioning is its reliance on real-time information from the cluster. The partitioner must continuously monitor each device's availability and status to ensure the workload is distributed to active, resourceful nodes. The leader node typically coordinates this process by invoking a DSE agent that evaluates the global computation-to-communication ratios of all nodes and predicts the latency of different configurations. This information is used to select the best partitioning mode—model-based, data-based, or hybrid—based on the DNN's characteristics and the cluster's current resource state. Figure 39 shows the layer-to-device bias of executing different layers of VGG-19 across four devices, including Jetson TX2, Jetson Nano, Raspberry Pi 4, and Raspberry Pi 5. The latency of most layers is lowest when executed on the Raspberry Pi 5; hence, a greedy scheduler will also prioritize this device for workload execution. However, in multiple run-time scenarios, the Raspberry Pi 5 may be unavailable due to previously running workloads or disconnections. Moreover, the communication cost of offloading the workload partition to a Raspberry Pi 5 may vary across devices. Hence, an intelligent global partitioning strategy should focus on computational capabilities, offloading costs, and node availability to make informed decisions that improve throughput and latency.

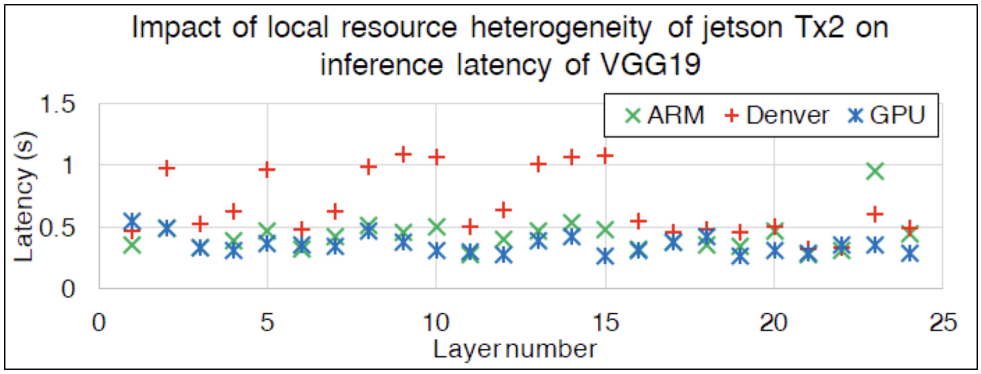

Figure 40. Partitioning of VGG-19 across the local computational clusters of Jetson TX2

6.2.2 Local workload distribution

Local workload distribution refers to the assignment and execution of workloads across heterogeneous local resources, including CPUs and GPUs, as discussed in chapters 3 and 4. Each computational cluster has distinct power, performance, energy, and memory characteristics. The goal is to exploit the intra-node heterogeneity to achieve optimized utilization of all available cores and minimize inference latency. By intelligently splitting and scheduling workloads locally, the node can execute assigned partitions more efficiently, reducing idle times and balancing compute-intensive and memory-intensive operations across processors.

The local partitioning decision is based on the computation-to-communication ratio, processor load, and type of operations in the DNN layers. For instance, convolutional and pooling layers are more GPU-friendly due to their high degree of parallelism. In contrast, fully connected layers and small-kernel operations may run more efficiently on CPUs. The local partitioner therefore determines the most suitable processor for each subtask, ensuring efficient core-level resource utilization. Figure 40 shows the execution of VGG-19 on different computational clusters of Jetson TX2, including ARM-CPU, Denver-CPU, and Pascal GPU. The layer-cluster bias indicates that layers are suitable for different clusters based on their types and the given MAC operations.

The local partitioning process typically includes three steps: analysis, partitioning, and scheduling. During the analysis stage, the run-time scheduler collects real-time communication overhead and computational performance of each cluster. This information is passed to a local DSE agent that evaluates possible partitioning modes, including layer-wise splitting or data-wise partitioning of the incoming workload. The agent computes the local computation rate (λ_k) and the local communication rate (μ_k) for each processor, forming a local resource vector $\psi\{\lambda, \mu\}$. These parameters are used to determine whether model or data partitioning is more beneficial in minimizing latency under current conditions. Figure 41 shows the different parti-

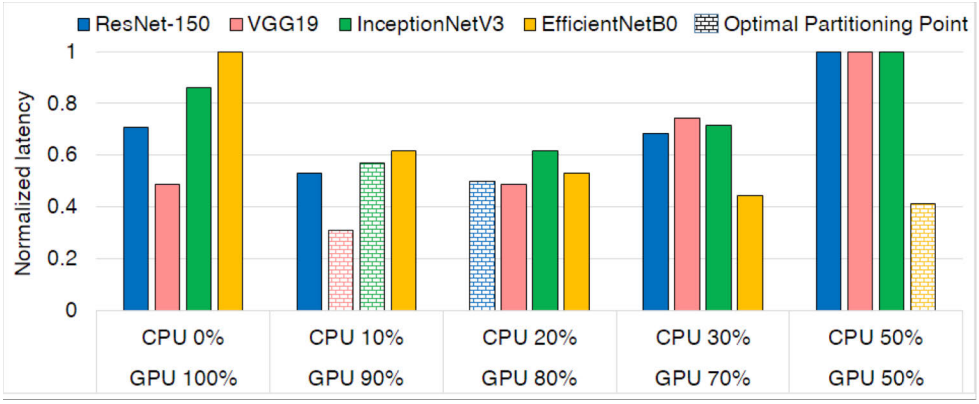

Figure 41. Partitioning of VGG-19 on Jetson TX2 against different partitioning points across local CPU and GPU clusters

tioning opportunities to split the workload of ResNet-50, VGG-19, InceptionNetv3, and EfficientNet across CPU and GPU clusters of Jetson TX2. The workload is distributed between the GPU and the CPU via model partitioning. Each model achieves its lowest latency at non-uniform partitioning points, as indicated by the shaded areas. An intelligent workload scheduling strategy should explore block sizes and partition counts to minimize latency.

Local partitioning significantly improves performance by avoiding the default Linux scheduling governor behavior, which often confines execution to a single processing unit, such as the GPU. By leveraging both the CPU and GPU simultaneously, local partitioning prevents resource underutilization and enhances throughput. Furthermore, it dynamically adapts to changing workloads and device conditions, enabling real-time optimization of the distribution of computation. This adaptability is particularly valuable in heterogeneous edge environments where performance characteristics may fluctuate due to temperature variations, power limitations, or concurrent workloads.

6.3 Existing strategies for workload partitioning

Current distributed edge inference approaches reduce inference latency by either splitting the workload along the DNN model structure or across the input data. Model partitioning techniques split the DNN into layer-wise blocks and execute them sequentially across multiple devices, enabling pipelined execution [12; 22; 63]. While effective for dense models with high computation-to-communication ratios, model partitioning is inherently constrained by the sequential nature of DNN architectures, as consecutive layers depend on intermediate activations from preceding layers. This limits the degree of parallelism that can be exploited across edge nodes, making per-

formance highly sensitive to partitioning points and pipeline balance.

Data partitioning techniques, in contrast, split the input data into smaller partitions that can be processed in parallel across multiple devices [11; 10; 17; 83; 95]. This approach improves spatial parallelism but introduces communication and synchronization overhead due to intermediate data exchange, particularly for DNN models with smaller input sizes or frequent feature map sharing. Hybrid partitioning strategies attempt to combine model and data partitioning to leverage the benefits of both approaches [23]. However, jointly coordinating model-wise sequential execution with data-wise parallel execution significantly increases scheduling complexity, especially in heterogeneous edge clusters with diverse compute and communication characteristics.

Most existing strategies make partitioning and workload-allocation decisions at the cluster level, treating each edge node as a single execution unit characterized by aggregate performance metrics [10; 11]. This abstraction simplifies scheduling but ignores the internal heterogeneity of edge nodes, which typically integrate multiple processing units such as CPUs, GPUs, and NPUs. As a result, the execution behavior within a node is decoupled from global partitioning decisions. After workload blocks are assigned to edge nodes, local execution is delegated to default Linux schedulers. These run-time systems independently map workloads to available processing units, often favoring a single accelerator, such as a GPU, by default [19; 99; 100]. This execution model does not account for variations in per-layer performance across processing units and fails to exploit parallelism within a node. Consequently, the effective compute capacity of an edge node is underestimated, and global workload allocation decisions are based on imprecise performance profiles.

Although several scheduling techniques explicitly account for core-level heterogeneity, they are primarily designed for inference on a single-edge device [22; 101; 14]. Extending such approaches to distributed inference requires coordinated decision-making across both cluster and node levels, along with mechanisms for inter-node data movement and control synchronization. Existing distributed inference strategies do not provide this level of orchestration, resulting in substantial inefficiencies in the joint management of sequential dependencies, parallelism, and heterogeneous resources.

6.4 HiDP framework

This dissertation presents *HiDP*, a hierarchical two-stage workload-splitting framework for distributed DNN inference across heterogeneous edge clusters. In this framework, multiple edge nodes are interconnected through wireless networks and collaboratively execute inference tasks in real time. Incoming DNN inference requests arrive asynchronously at a local node, where *HiDP* performs a hierarchical partitioning process to minimize end-to-end inference latency. As illustrated in Fig-

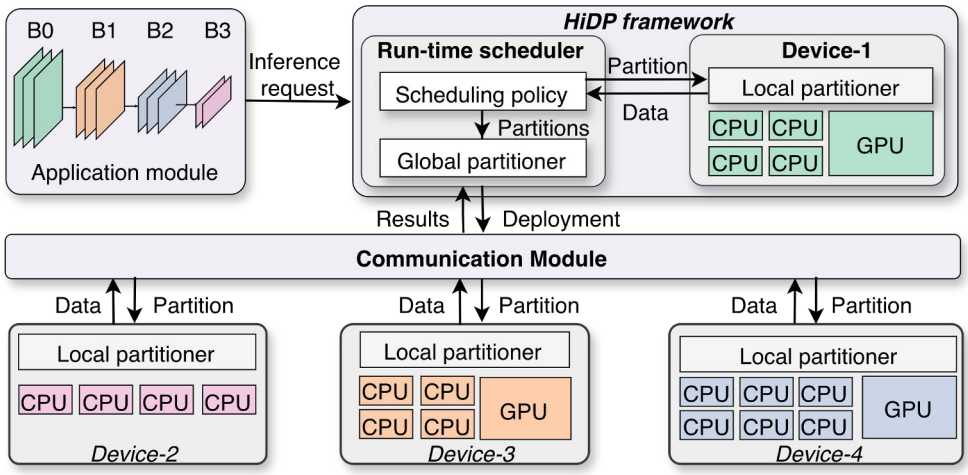

Figure 42. The proposed HiDP framework showing the inference of a DNN workload through local and global partitioning

ure 42, *HiDP* architecture includes the *Application Module*, *Communication Module*, *Global Partitioner*, *Local Partitioner*, and a *Run-time Scheduler*.

When a DNN inference request is received by the *Application Module* on Device-1, it is forwarded to the *Run-time Scheduler*, which orchestrates the workload distribution and execution across the cluster. The scheduler continuously monitors the availability and performance of all nodes and interacts with the *Global Partitioner* to determine the most efficient global partitioning point. The *Global Partitioner* leverages a DSE agent to explore potential partitioning modes and select the optimal configuration based on the computation-to-communication characteristics of the participating nodes. Once the global partitioning decision is made, the workload is distributed to other devices in the cluster through the *Communication Module*, which handles data exchange and synchronization between nodes.

Following the global distribution, the *Global Partitioner* forwards the local portion of the workload to the *Run-time Scheduler* for intra-node execution. At this stage, the scheduler invokes the *Local Partitioner*, which employs a *DSE* agent to determine the most suitable local partitioning mode. The *Local Partitioner* performs fine-grained workload splitting that accounts for the heterogeneity of local hardware resources, such as CPUs and GPUs, to ensure balanced utilization. For example, in a configuration with four CPUs and one GPU, it divides the workload proportionally to match the compute efficiency of each unit. Upon completion, the *Run-time Scheduler* collects both local and global inference results through the *Communication Module* and merges them to generate the final prediction output. This hierarchical coordination of global and local partitioning enables *HiDP* to achieve efficient distributed inference with significantly reduced latency and optimized resource utilization across

Table 6. Technical specifications of the evaluation setup

Device	CPU	GPU	DRAM
Jetson Orin NX	8x ARM Cortex-A78	1024-core Ampere	8 GB
Jetson TX2	2x Denver-2, 4x ARM Cortex-A57	256-core Pascal	8 GB
Jetson Nano	4x ARM Cortex-A57	128-core Maxwell	4 GB
Raspberry Pi 5	2x ARM Cortex-A76	VideoCore VII	4 GB
Raspberry Pi 4	2x ARM Cortex-A72	VideoCore VI	4 GB

heterogeneous edge platforms.

6.5 Experimental evaluation and benchmarks

The considered heterogeneous edge cluster comprises commercial edge platforms (listed in Table 6). *HiDP* monitors the run-time power consumption of the Jetson boards using the on-board power sensors. In contrast, the Raspberry Pi boards' power consumption is monitored using an external shunt resistor. The framework was evaluated on ResNet-152, EfficientNet-B0, VGG-19, and Inception-NetV3, which are widely used DNN models in mobile vision applications. These models were implemented using the TensorFlow library [19], with input image sizes of 224 × 224 and 299 × 299. Their top layers were modified to accommodate variable input sizes, thereby enabling data partitioning. The *HiDP* middleware, implemented in Python and comprising approximately 400 lines of code, runs on Ubuntu 18.04 on all devices to provide the necessary software and hardware interfaces. CGroups libraries were used to bind workloads to the desired number of CPU cores, while GPU execution used CUDA on NVIDIA boards and OpenGL on Raspberry Pi boards. All devices are connected via an 80 Mbps wireless network using a POSIX-based client-server architecture, with multi-threaded server operations that allow leader nodes to dynamically communicate with available nodes. The exploration overhead for both global and local partitioning is approximately 15 ms in the experiments. For evaluation, *HiDP* was compared against three SoA partitioning strategies: *MoDNN* [11], which partitions and distributes input data proportionally among available edge nodes; *OmniBoost* [12], which determines optimal model partition points using Monte-Carlo search and pipelines DNN inference over both CPU and GPU; and *DisNet* [23], which uses heuristic-based hybrid data and model partitioning. For *MoDNN*, the data partitioning module was implemented using the *HiDP* framework. In contrast, the throughput estimator of *OmniBoost* was implemented with the Gymnasium library [94] and trained on the target workloads. The hybrid partitioning of *DisNet* was implemented using the data- and model-partitioning algorithm of *HiDP*.

Experimental results Figure 43 presents the comprehensive evaluation of the HiDP framework against SoA distributed inference strategies, including DisNet, OmniBoost, and MoDNN, across four widely-used DNN models, including Efficient-

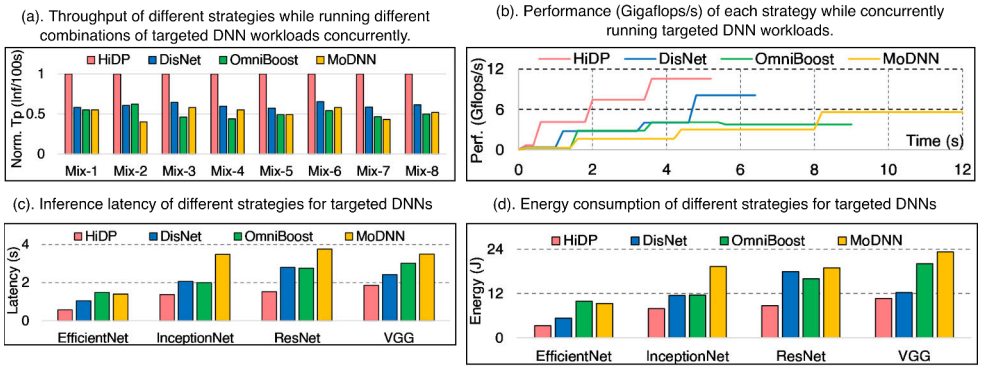

Figure 43. Evaluation results for executing eight different workload mixes across four different strategies

Table 7. Workload mixes used for throughput evaluation

Mix	DNN Models
Mix-1	EfficientNetB0 + InceptionNetV3
Mix-2	EfficientNetB0 + ResNet152
Mix-3	EfficientNetB0 + VGG-19
Mix-4	ResNet152 + InceptionNetV3
Mix-5	EfficientNetB0 + InceptionNetV3 + ResNet152
Mix-6	EfficientNetB0 + InceptionNetV3 + VGG-19
Mix-7	EfficientNetB0 + ResNet152 + VGG-19
Mix-8	InceptionNetV3 + ResNet152 + VGG-19

NetB0, InceptionNetV3, ResNet152, and VGG-19. As illustrated in Figure 43(c), HiDP achieves the lowest inference latency for all workloads, demonstrating up to 61%, 61%, 59%, and 49% latency reduction for EfficientNet, InceptionNet, ResNet, and VGG, respectively, compared to other strategies. On average, HiDP achieves latency reductions of 37%, 44%, and 56% relative to the DisNet, OmniBoost, and MoDNN strategies, respectively. The hierarchical partitioning strategy employed by HiDP jointly optimizes global-level DNN block creation and workload distribution, followed by local-level fine-grained partitioning and workload scheduling, yielding significantly lower latency than strategies that rely solely on global partitioning. The energy efficiency gains shown in Figure 43(d) mirror these latency improvements, with HiDP consuming up to 67%, 59%, 54%, and 54% lower energy for EfficientNet, InceptionNet, ResNet, and VGG, respectively. On average, HiDP consumes 33%, 48%, and 58% less energy than the DisNet, OmniBoost, and MoDNN strategies, respectively.

The adaptability of HiDP under dynamic workload conditions is demonstrated in Figures 43(a) and 43(b). Figure 43(b) shows the performance (Gigaflops per second) during concurrent execution of multiple DNNs, where HiDP consistently delivers

the highest performance throughout the execution cycle. The lower latency achieved by HiDP frees up edge node resources more quickly, enabling efficient servicing of subsequent inference requests. HiDP completes inference across all models in 5 seconds, achieving 39%, 54%, and 56% higher performance than DisNet, OmniBoost, and MoDNN, respectively. We evaluated each strategy across eight workload mixes as shown in Table 7. The throughput evaluation in Figure 43(a) across different workload mixes demonstrates that HiDP achieves significantly higher throughput, with improvements up to 150% in Mix-2 and 56% on average across all mixes. This superior performance stems from HiDP's ability to dynamically select between data and model partitioning based on DNN characteristics, enabling low-latency inference across different workload combinations, whereas other strategies remain confined to specific partitioning configurations.

7 Conclusion and Future Work

7.1 Thesis summary

This dissertation presented a comprehensive investigation into the design, orchestration, and optimization of inference across multiple ML frameworks on heterogeneous, distributed edge platforms. The thesis begins with the fundamental principles of multi-application RTM on mobile HMPs, progresses to multi-DNN orchestration, compound AI workloads, and ultimately, hierarchical distributed inference strategies. The work systematically addressed the challenge of meeting real-time performance, energy efficiency, and accuracy requirements while operating under limited, heterogeneous resources. The hierarchical structure of the thesis allowed each stage to contribute incrementally to a unified vision of intelligent, low-latency, and scalable DNN execution at the edge.

7.1.1 Broader Implications and Unifying Perspective

This dissertation demonstrates that efficient edge AI inference cannot be addressed by isolated optimizations on single devices, single models, or single-level schedulers. Instead, it requires a holistic view of the computation pipeline that spans heterogeneous hardware, diverse model types, dynamic workloads, and distributed topologies. A common principle emerges from the individual contributions: robust edge intelligence requires a nuanced run-time context that adaptively coordinates accuracy, performance, and energy across the system, workload, and application scenarios. This principle shapes the design of the run-time resource management techniques, the multi-DNN and compound AI schedulers, and the distributed inference frameworks proposed in this work.

Each publication points toward a unified direction that edge AI systems require end-to-end orchestration stacks that combine model-level, system-level, and network-level intelligence. Accuracy, latency, and energy cannot be treated as individual objectives; they must be jointly optimized. The frameworks developed in this dissertation provide foundational building blocks for an efficient run-time for an edge AI system. They demonstrate that, with the right abstractions, monitoring infrastructure, and adaptive policies, heterogeneous edge platforms can deliver reliable, scalable, and energy-efficient AI services even under challenging, unpredictable operating conditions.

In the broader context of edge computing, these results emphasize that the future of AI deployment lies not only in designing more efficient accelerators or compressed models but also in creating run-time intelligence capable of orchestrating complex AI ecosystems. As AI services increasingly move closer to users and interact with real-time sensor data, the ability to schedule, adapt, migrate, and reconfigure inference pipelines will become a defining capability for next-generation edge platforms. This dissertation contributes to that vision by presenting principled methods and practical systems that push the boundaries of what heterogeneous edge devices can achieve under real-world constraints.

7.2 Open direction for future research

Despite the promising outcomes of this thesis, there are still some open challenges left to explore in the future:

- **Exploiting accelerators:** The HiDP framework primarily targets CPU and GPU clusters in a distributed inference mechanism, while there are still computational opportunities left to exploit NPUs, DLAs, or custom ASIC accelerators for latency and energy improvements.
- **Node disconnections:** Handling the unavailability of edge cluster nodes at run-time due to errors and network unavailability is still an open challenge for avoiding inference bottlenecks and data loss.
- **Communication overheads:** This thesis is entirely focused on improving computational usability of available resources, while the significant communication overheads in a distributed setup can still be improved in future works.
- **Edge-cloud collaboration:** This research work is entirely focused on edge-based inference, but the same technique can be applied to an edge-cloud paradigm, exploring opportunities for optimizing inference of dense on-device large-language models.

Overall, this dissertation provides the theoretical and practical foundations for scalable, adaptive deep learning inference in heterogeneous, distributed edge environments. The proposed frameworks collectively bridge the gap among hardware heterogeneity, AI workload complexity, and real-time responsiveness, thereby establishing a strong foundation for the next generation of intelligent edge systems.

List of References

[1] Jingtao Ding, Yunke Zhang, Yu Shang, Yuheng Zhang, Zefang Zong, Jie Feng, Yuan Yuan, Hongyuan Su, Nian Li, Nicholas Sukiennik, et al. Understanding world or predicting future? a comprehensive survey of world models. *ACM Computing Surveys*, 58(3):1–38, 2025.

[2] Grzegorz Malczyk, Mihir Kulkarni, and Kostas Alexis. Semantically-driven deep reinforcement learning for inspection path planning. *IEEE Robotics and Automation Letters*, 10(7):7206–7213, 2025. doi: 10.1109/LRA.2025.3575331.

[3] Dominic Maggio, Yun Chang, Nathan Hughes, Matthew Trang, Dan Griffith, Carlyn Dougherty, Eric Cristofalo, Lukas Schmid, and Luca Carlone. Clio: Real-time task-driven open-set 3d scene graphs. *IEEE Robotics and Automation Letters*, 9(10):8921–8928, 2024. doi: 10.1109/LRA.2024.3451395.

[4] Apple Inc. Apple intelligence, 2024. URL `https://www.apple.com/apple-intelligence/`. Accessed: 2025-04-21.

[5] Apple Inc. Apple vision pro: Spatial computing device, 2024. URL `https://www.apple.com/apple-vision-pro/`. Accessed: 2025-07-10.

[6] Meta Platforms Inc. Ray-ban meta ai glasses. `https://www.ray-ban.com/usa/ray-ban-meta-ai-glasses`, 2023. Accessed: 2025-05-05.

[7] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound ai systems. `https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/`, 2024.

[8] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras. Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 27(12):2842–2854, 2019.

[9] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett. AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2206–2217, 2020.

[10] 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. ISSN 02780070. doi: 10.1109/TCAD.2018.2858384.

[11] Jiachen Mao et al. MoDNN: Local distributed mobile computing system for Deep Neural Network. *Proc. of Design, Automation and Test in Europe, DATE*, pages 1396–1401, 2017. doi: 10.23919/DATE.2017.7927211.

[12] A. Karatzas and I. Anagnostopoulos. OmniBoost: Boosting Throughput of Heterogeneous Embedded Devices under Multi-DNN Workload. In *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, pages 1–6, 2023.

[13] Andreas Karatzas and Iraklis Anagnostopoulos. Mapformer: Attention-based multi-dnn manager for throughout & power co-optimization on embedded devices. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710773.

[14] Joo 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.

[15] Ismet Dagli and Mehmet E. Belviranli. Shared memory-contention-aware concurrent dnn execution for diversely heterogeneous system-on-chips. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, page 243–256. ACM, February 2024.

[16] L. Zhou et al. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In *IEEE SEC*, page 195–208, 2019.

[17] K Choi et al. Legion: Tailoring grouped neural execution considering heterogeneity on multiple edge devices. In *IEEE Int. Conf. on Computer Design (ICCD)*, pages 383–390, 2021. doi: 10.1109/ICCD53106.2021.00067.

[18] NVIDIA. *NVIDIA TensorRT Documentation*, 2024. URL `https://docs.nvidia.com/deeplearning/tensorrt/`. Accessed: 2025-06-17.

[19] Ashish Agarwal Abadi et al. Tensorflow. `https://www.tensorflow.org/`, 2015.

[20] PyTorch Team. Pytorch installation guide. `https://pytorch.org/get-started/locally/`, 2024. Accessed: 2025-07-10.

[21] Tencent. ncnn: High-performance neural network inference framework. `https://github.com/Tencent/ncnn`, 2024. Accessed: 2025-07-10.

[22] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra. High-Throughput CNN Inference on Embedded ARM Big.LITTLE Multicore Processors. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2254–2267, 2019.

[23] Eric Samikwa et al. Disnet: Distributed micro-split deep learning in heterogeneous dynamic iot. *IEEE Internet of Things Journal*, 11(4):6199–6216, 2024. doi: 10.1109/JIOT.2023.3313514.

[24] Xiaotian Guo, Quan Jiang, Yixian Shen, Andy D. Pimentel, and Todor Stefanov. Easter: Learning to split transformers at the edge robustly. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3626–3637, 2024.

[25] S. Isuwa, S. Dey, A. P. Ortega, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett. QUAREM: Maximising QoE through Adaptive Resource Management in Mobile MPSoC Platforms. *ACM Trans. on Embedded Computing Systems (TECS)*, 2022.

[26] Yanchun Liu, Hongxue Qu, Shuang Chen, and Xuejun Feng. Energy efficient task scheduling for heterogeneous multicore processors in edge computing. *Scientific Reports*, 15(1):11819, 2025.

[27] Anil Kanduri et al. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *ACM Design Automation Conference (DAC)*, pages 1–6, 2018. doi: 10.1109/DAC.2018.8465797.

[28] D. Angioletti, F. Bertani, B. Bolchini, F. Cerizzi, and A. Miele. A Runtime Resource Management Policy for OpenCL Workloads on Heterogeneous Multicores. In *Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1385–1390, 2019.

[29] ARM Ltd. big.LITTLE Technology: The Future of Mobile. White paper, ARM Limited, Cambridge, United Kingdom, 2013.

[30] Zain Taufique, Aman Vyas, Antonio Miele, Pasi Liljeberg, and Anil Kanduri. HiDP: Hierarchical DNN Partitioning for Distributed Inference on Heterogeneous Edge Platforms. *Proc. Design, Automation and Test in Europe Conf. (DATE)*, pages 1–6, 2025.

[31] Kisoo K. Yu, D. Han, C. Youn, S. Hwang, and J. Lee. Power-aware task scheduling for big.LITTLE mobile processor. In *Proc. of Intl. SoC Design Conf. (ISOCC)*, pages 208–212, 2013.

[32] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *Proc. of Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

[33] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras. DyPO: Dynamic Pareto-Optimal Configuration Selection for Heterogeneous MpSoCs. *ACM Trans. on Embedded Computing Systems*, 16(5s), 2017.

[34] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt. Approximation-aware Coordinated Power/Performance Management for Heterogeneous Multi-cores. In *Proc. of Design Automation Conf. (DAC)*, pages 68:1–68:6, 2018.

[35] Hardkernel co. ODROID. http://www.hardkernel.com, 2015. Accessed: 2022-09-01.

[36] Martin Rapp, Mohammed Bakr Sikal, Heba Khdr, and Jörg Henkel. Smartboost: Lightweight ml-driven boosting for thermally-constrained many-core processors. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 265–270. IEEE, 2021.

[37] Ujjwal Gupta, Manoj Babu, Raid Ayoub, Michael Kishinevsky, Francesco Paterna, and Umit Y Ogras. Staff: Online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[38] E. Shamsa, A. Kanduri, P. Liljeberg, and A. M. Rahmani. Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms. *IEEE Trans. on Computers*, 71(4): 743–755, 2021.

[39] Paul Menage (based on CPUSETS) et al. *Control Groups — The Linux Kernel documentation (cgroup v1)*. The Linux Kernel Organization, 2021. URL https://www.kernel.org/doc/html/v5.15/admin-guide/cgroup-v1/cgroups.html. Accessed: 2025-11-06.

[40] A. Pathania, J. Qing, A. Prakash, and T. Mitra. Integrated CPU-GPU power management for 3D mobile games. In *Proc. Design Automation Conf. (DAC)*, pages 1–6, 2014.

[41] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel. Power Density-Aware Resource Management for Heterogeneous Tiled Multicores. *IEEE Trans. on Computers*, 66(3):488–501, 2017.

[42] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, and N. Dutt. Approximation knob: Power Capping meets energy efficiency. In *Proc. of ACM/IEEE Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 1–8, 2016.

[43] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 618–623, 2015.

[44] S. Conoci, P. Di Sanzo, B. Ciciani, and F. Quaglia. Adaptive Performance Optimization Under Power Constraint in Multi-thread Applications with Diverse Scalability. In *Proc. of ACM/SPEC Intl. Conf. on Performance Engineering (ICPE)*, pages 16–27, 2018.

[45] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. Workload-aware Power Optimization Strategy for Asymmetric Multiprocessors. In *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*, pages 531–534, 2016.

[46] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev. Power–Aware Performance Adaptation of Concurrent Applications in Heterogeneous Many-Core Systems. In *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED)*, pages 368–373, 2016.

[47] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. In *Proc. of ACM Symp. and European Conf. on Foundations of Software Engineering (ESEC/FSE)*, pages 124–134, 2011.

[48] D. Palomino, M. Shafique, A. Susin, and J. Henkel. Thermal optimization using adaptive approximate computing for video coding. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1207–1212, 2016.

[49] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique. Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1384–1389, 2017.

[50] S. Chakraborty, S. Saha, M. Själander, and K. Mcdonald-Maier. Prepare: Power Aware Approximate Real-time Task Scheduling for Energy-Adaptive QoS Maximization. *ACM Trans. on Embedded Computing Systems (TECS)*, 20(5s):1–25, 2021.

[51] ASUS. Tinker edge r. https://tinker-board.asus.com/series/tinker-edge-r.html, 2024.

[52] R. R. Curtin et al. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14(1):801–805, 2013.

[53] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. of IEEE Intl. Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009.

[54] Zain Taufique et al. Approximate feature extraction for low power epileptic seizure prediction in wearable devices. In *IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7, 2021. doi: 10.1109/NorCAS53631.2021.9599870.

[55] A. Arranz-Gimon, A. Zorita-Lamadrid, D. Morinigo-Sotelo, and O. Duque-Perez. A Review of Total Harmonic Distortion Factors for the Measurement of Harmonic and Interharmonic Pollution in Modern Power Systems. *Energies*, 14(20), 2021.

[56] Lixiang Han, Zimu Zhou, and Zhenjiang Li. Pantheon: Preemptible multi-dnn inference on mobile edge gpus. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, pages 465–478, 2024.

[57] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. *Neurocomputing*, 485:297–320, 2022.

[58] Raghubir Singh and Sukhpal Singh Gill. Edge ai: a survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023.

[59] Zain Taufique, Aman Vyas, Antonio Miele, Pasi Liljeberg, and Anil Kanduri. Twill: Scheduling compound ai systems on heterogeneous mobile edge platforms. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2025. doi: 10.1109/ICCAD66269.2025.11240767.

[60] Andreas Karatzas, Dimitrios Stamoulis, and Iraklis Anagnostopoulos. Rankmap: Priority-aware multi-dnn manager for heterogeneous embedded devices, 2024.

[61] Zain Taufique, Aman Vyas, Antonio Miele, Pasi Liljeberg, and Anil Kanduri. Tango: Low latency multi-dnn inference on heterogeneous edge platforms. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, pages 300–307, 2024. doi: 10.1109/ICCD63220.2024.00053.

[62] S. Minakova et al. Scenario based run-time switching for adaptive cnn-based applications at the edge. *ACM Trans. on Embedded Computing Systems (TECS)*, 21(2):1–33, 2022.

[63] Y. Wu, Y. Gong, Z. Zhan, G. Yuan, Y. Li, Qi Wang, C. Wu, and Y. Wang. MOC: Multi-Objective Mobile CPU-GPU Co-Optimization for Power-Efficient DNN Inference. In *Proc. of ACM/IEEE Intl. Conf. on Computer Aided Design (ICCAD)*, pages 1–10, 2023.

[64] Chengdong Lin, Kun Wang, Zhenjiang Li, and Yu Pu. A workload-aware dvfs robust to concurrent tasks for mobile devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399906.

[65] Ioannis Panopoulos, Stylianos Venieris, and Iakovos Venieris. Carin: Constraint-aware and responsive inference on heterogeneous devices for single- and multi-dnn workloads. *ACM Trans. Embed. Comput. Syst.*, 23(4), June 2024. ISSN 1539-9087.

[66] Zain Taufique et al. Adaptive workload distribution for accuracy-aware dnn inference on collaborative edge platforms. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024. doi: 10.1109/ASP-DAC58780.2024.10473987.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[68] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[69] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017. doi: 10.1109/TPAMI.2016.2572683.

[70] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[71] Young Geun Kim et al. Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. *Proc. of Int. Symp. on Microarchitecture, MICRO*, pages 1082–1096, 2020. ISSN 10724451. doi: 10.1109/MICRO50266.2020.00090.

[72] NVIDIA. Jetson tx2 module, 2024. URL https://developer.nvidia.com/embedded/jetson-tx2.

[73] Torchvision. https://pytorch.org/vision/stable/index.html, 2024. Accessed: 2025-07-10.

[74] Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei A Zaharia, and James Y Zou. Are more llm calls all you need? towards the scaling properties of compound ai systems. *Advances in Neural Information Processing Systems*, 37:45767–45790, 2024.

[75] Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Matei Zaharia, James Zou, and Ion Stoica. Optimizing model selection for compound ai systems. *arXiv preprint arXiv:2502.14815*, 2025.

[76] Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans? *arXiv preprint arXiv:2310.08118*, 2023.

[77] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[78] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. Drive as you speak: Enabling human-like interaction with large language models in autonomous vehicles. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 902–909, 2024.

[79] Harrison Chase. Langchain, 2022. URL https://github.com/langchain-ai/langchain. Release date: 2022-10-17.

[80] Langbase. Langbase: AI-Powered Multilingual Database, 2025. URL https://langbase.com/.

[81] NVIDIA Corporation. Working with dla, 2025. URL https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/work-with-dla.html.

[82] NVIDIA. Nvidia jetson Orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2024. Accessed: 2025-07-10.

[83] Xiaotian Guo et al. Automated exploration and implementation of distributed cnn inference at the edge. *IEEE Internet of Things Journal*, 10(7):5843–5858, April 2023. ISSN 2327-4662. doi: 10.1109/JIOT.2023.3237572.

[84] Qualcomm. Qualcomm launches its next-generation xr and ar platforms, 2023. URL https://www.qualcomm.com/news/releases/2023/09/qualcomm-launches-its-next-generation-xr-and-ar-platforms--enabl. Accessed: 2025-04-08.

[85] Meta Platforms Inc. Meta ray-ban smart glasses: Next generation smart eyewear. *Meta Technology Review*, 2023. URL https://www.ray-ban.com/usa/ray-ban-meta-ai-glasses.

[86] Hung-Yang Chang, Seyyed Hasan Mozafari, Cheng Chen, James J. Clark, Brett H. Meyer, and Warren J. Gross. Pipebert: High-throughput bert inference for arm big.little multi-core processors. *J. Signal Process. Syst.*, 95:877–894, October 2022.

[87] ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021.

[88] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[89] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, 2019.

[90] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[91] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[92] Hugging Face. Hugging face: The ai community building the future of machine learning, 2025. URL `https://huggingface.co/`. Accessed: 2025-04-18.

[93] Ollama Inc. Ollama: Run large language models locally, 2025. URL `https://ollama.com/`. Accessed: 2025-04-18.

[94] Mark Towers et al. Gymnasium, 2023. URL `https://zenodo.org/record/8127025`. Accessed: 2025-07-10.

[95] Yakun Huang et al. Enabling DNN Acceleration with Data and Model Parallelization over Ubiquitous End Devices. *IEEE Internet of Things Journal*, 2021. ISSN 23274662. doi: 10.1109/JIOT.2021.3112715.

[96] Hans Jakob Damsgaard, Antoine Grenier, Dewant Katare, Zain Taufique, Salar Shakibhamedan, Tiago Troccoli, Georgios Chatzitsompanis, Anil Kanduri, Aleksandr Ometov, Aaron Yi Ding, et al. Adaptive approximate computing in edge ai and iot applications: A review. *Journal of Systems Architecture*, page 103114, 2024.

[97] Feng Xue et al. EdgeLD: Locally Distributed Deep Learning Inference on Edge Device Clusters. *IEEE CHPCC*, pages 613–619, 2020. doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00078.

[98] Sina Shahhosseini et al. Online learning for orchestration of inference in multi-user end-edge-cloud networks. *ACM TECS*, dec 2022. ISSN 1539-9087. doi: 10.1145/3520129.

[99] TensorFlow Developers. *TensorFlow Guide: GPU support and Manual Device Placement*, 2023. URL `www.tensorflow.org/guide/gpu/#manual_device_placement`.

[100] Princeton Research Computing. *TensorFlow on Princeton Research Computing Clusters*, 2023.

[101] Ehsan Aghapour et al. Arm-co-up: Arm co operative u tilization of p rocessors. *ACM Transactions on Design Automation of Electronic Systems*, 2024.