NPSIM: A Tick-Accurate CPU–NPU Simulator for Scheduling Analysis of CNN Applications

Mourad DRIDI Görkem SALMAN Burak BASTUG Muhammed Furkan BASIBÜYÜK
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

***Abstract*— Applied Artificial Intelligence (AI) is increasingly used in emerging autonomous systems such as self-driving vehicles, drones, and robotics, where perception and decision-making rely on deep learning models, notably Convolutional Neural Networks (CNNs). These applications operate under strict timing constraints, as meeting inference deadlines is essential to ensure efficiency, safety, and robustness in dynamic environments. To accelerate inference on embedded platforms, GPUs are progressively being replaced by Neural Processing Units (NPUs), which are better suited to resource-constrained environments. However, NPU architectures are typically throughput-oriented, rely on non-preemptive execution, and lack native real-time guarantees. As a result, their timing behavior is difficult to predict, significantly complicating their integration into real-time autonomous systems and the enforcement of end-to-end deadline constraints. Moreover, existing real-time simulation and scheduling analysis tools fail to accurately capture the specific execution characteristics of CPU–NPU architectures, particularly the layer-level execution of CNNs and the possible parallel execution of layers belonging to different CNNs. Consequently, system designers lack appropriate tools to explore scheduling strategies and assess system-level schedulability under realistic architectural constraints. To address these challenges, we propose npsim, a tick-accurate CPU–NPU simulator dedicated to the scheduling analysis of CNN-based applications. npsim models the detailed timing behavior of heterogeneous AI workloads, including non-preemptive NPU execution, data-transfer delays, and layer-level CNN execution. It provides a realistic environment for exploring scheduling policies and analyzing the trade-off between CNN result accuracy and system schedulability. Its lightweight design further enables easy integration into the development workflow of commercial NPUs, such as the NXP Neutron NPU.**

I. Introduction

Over the past decade, Artificial Intelligence (AI) has become a key component of emerging autonomous systems such as self-driving vehicles, drones, robotics, and intelligent IoT platforms. In these systems, AI-based perception and decision-making functions are commonly implemented using Convolutional Neural Networks (CNNs). Since the output of these AI components directly affects control and actuation, delayed or unpredictable inference can, in safety-critical scenarios such as autonomous driving, lead to incorrect system behavior with severe or even catastrophic consequences [7]. Many autonomous systems are therefore characterized by hard real-time constraints, where AI inference tasks are required to complete within specified deadlines to ensure correct system behavior. In such systems, correctness depends not only on the logical results of computations, but also on the time at which these results are produced [10], [5].

To meet the computational demands of CNN inference under strict energy and resource constraints, embedded platforms increasingly adopt Neural Processing Units (NPUs), progressively replacing GPUs. NPUs are designed to maximize throughput and energy efficiency, which makes them attractive for embedded AI [8]. However, current NPU architectures are typically throughput-oriented, rely on non-preemptive execution, and provide limited or no real-time guarantees [7]. As a result, the timing behavior of AI workloads executing on NPUs is difficult to predict, which complicates schedulability analysis and the integration of AI inference tasks into real-time autonomous systems deployed on CPU–NPU heterogeneous architectures [9]. In addition, the execution time of CNN layers on NPUs strongly depends on the selected numerical precision (e.g., INT8, FP16, FP32). Higher precision generally leads to increased execution time and resource usage, while lower precision improves performance at the cost of reduced inference accuracy. This introduces a fundamental trade-off between CNN result accuracy and the ability to meet real-time deadlines, making scheduling decisions tightly coupled with precision selection [9]. Accurate scheduling analysis of CNN execution on CPU–NPU heterogeneous architectures is further challenged by several factors, including layer-level execution, data-transfer delays between processing units and the possible parallel execution of layers belonging to different CNNs. In addition, existing real-time simulators and analysis frameworks often ignore these NPU-specific characteristics or rely on overly abstract models, making them unsuitable for realistic system-level schedulability analysis of autonomous systems.

To address these challenges, we introduce npsim, a tick-accurate CPU–NPU simulator dedicated to the scheduling analysis of CNN-based applications deployed on heterogeneous architectures. npsim models the detailed temporal behavior of AI workloads, including non-preemptive NPU execution, data-transfer delays and layer-level CNN execution. It enables designers to explore scheduling policies, assess whether each task meets its deadline, and analyze the trade-off between CNN result accuracy and system schedulability on modern NPU-based platforms. The proposed simulator has been extensively validated through a combination of unit tests, functional tests, and consistency checks to ensure the correctness of its execution and scheduling models. npsim implements several classical real-time scheduling algorithms commonly used in CPU–NPU architectures, including Earliest Deadline First (EDF), Rate Monotonic (RM),

and Fixed-Priority (FP) scheduling, enabling comparative evaluation under different scheduling policies. We further conducted a series of experimental evaluations to assess the computational overhead and scalability of the simulator. The simulation results were compared against measurements obtained on existing NPU-based platforms, such as MCX-N. These evaluations demonstrate that npsim can efficiently analyze realistic CPU–NPU configurations while maintaining tick-accurate timing fidelity. Overall, npsim provides system designers with a practical tool to dimension AI-enabled real-time systems, explore scheduling and precision-selection strategies, and reason about timing guarantees early in the design process. This paper is organized as follows. Section II introduces real-time systems, CNN applications, and CPU–NPU execution models. Section III presents the design of the *npsim* simulator. Section IV describes the implementation of *npsim*. Section V presents the evaluation of the proposed simulator. Finally, related works and conclusions are discussed in Sections VI and VII, respectively.

II. Background

A. Real-Time Systems and Scheduling

Real-time systems are characterized by timing constraints that must be respected to guarantee correct system behavior. Typically, each task is defined by a set of timing attributes, including its worst-case execution time (WCET), period, release time, deadline, and other parameters depending on the considered task model. A real-time system is considered schedulable if all task instances meet their deadlines (i.e., complete their execution before their corresponding deadlines) under a given scheduling policy [10], [5].

According to the criticality of timing requirements, real-time systems are commonly classified into *hard* and *soft* real-time systems. In hard real-time systems, missing a deadline may result in severe consequences or complete system failure. In contrast, soft real-time systems can tolerate occasional deadline misses without causing system failure, although such violations may lead to a degradation of service quality. Typical examples of hard real-time systems include autonomous driving functions, flight control tasks in drone systems, and obstacle detection in autonomous vehicles, whereas soft real-time systems are commonly found in applications such as video conferencing or multimedia [10], [5]. A variety of classical real-time scheduling algorithms have been proposed in the literature, including fixed-priority approaches such as Rate Monotonic (RM), as well as dynamic-priority algorithms such as Earliest Deadline First (EDF). These algorithms have been extensively analyzed for CPU-based platforms and are widely adopted in embedded and safety-critical real-time systems [5].

B. Lightweight CNNs

Convolutional Neural Networks (CNNs) are a class of artificial intelligence models widely used in autonomous and intelligent systems. A CNN is composed of multiple layers, where each layer consists of a set of computational nodes performing specific operations. CNN-based processing generally involves two main phases: *training*, during which model parameters are learned from large datasets, and *inference*, where the trained model performs predictions on previously unseen data in real time [7].

In this work, we focus on the inference phase, which is typically deployed on resource-constrained embedded platforms, such as autonomous drones or mobile robotic systems. To satisfy the stringent computational and timing constraints of such platforms, several lightweight CNN architectures have been specifically designed or optimized for efficient execution. Representative examples include MobileNet, Tiny YOLO, and SimpleNet, which enable real-time applications such as object detection, classification, and segmentation on embedded systems [8]. From a real-time perspective, the execution of a CNN can be modeled as a pipeline of dependent tasks, where each task corresponds to the execution of a single network layer. Consequently, the end-to-end latency of a CNN depends not only on the execution time of individual layers, but also on the scheduling decisions and resource contention arising from the sharing of computing resources [10].

C. Heterogeneous CPU–NPU Architectures

To meet performance and energy-efficiency requirements, modern embedded platforms increasingly rely on heterogeneous architectures combining general-purpose CPUs with NPUs. In such systems, classical real-time tasks are typically executed on the CPU, while CNN inference workloads are offloaded to the NPU [8]. NPUs often exhibit non-preemptive execution semantics and possible parallel execution, which introduce new challenges for real-time scheduling and analysis. The coexistence of CPU and NPU workloads leads to complex interactions, including resource contention, blocking effects, and trade-offs between throughput, numerical precision, and timing guarantees. These characteristics motivate the need for system-level tools capable of accurately modeling CPU–NPU execution and evaluating schedulability under different scheduling and system configurations, as addressed in this work.

III. Overview of the Proposed Simulator: *npsim*

This section presents *npsim*, a system-level simulator designed to analyze the schedulability of real-time applications integrating artificial intelligence tasks on heterogeneous CPU–NPU architectures. The main objective of *npsim* is to evaluate whether real-time constraints can be satisfied when classical real-time tasks and AI inference tasks execute concurrently on shared computing resources.

Unlike performance-oriented or micro-architectural simulators, *npsim* focuses on temporal correctness at the system level. It enables designers to explore trade-offs between task mapping, scheduling policies, numerical precision of CNN execution, and schedulability guaranties. In the following, we describe the input and output models of the simulator, before detailing its execution workflow, supported scheduling policies, and execution semantics.

A. Global Workflow

1) Simulation Input: As illustrated in Fig. 2, *npsim* relies on a textual system description provided through an input file (`input.txt`). This file specifies the application task model, the underlying hardware architecture, and the mapping of tasks to computing resources. In this work, applications are modeled as a set of dependent periodic tasks, inspired by the task model proposed in [10] and extended to capture AI workloads executed on CPU–NPU architectures. In this work, we consider two types of tasks:

Non-CNN tasks, corresponding to classical real-time tasks executed on the CPU (e.g., sensing, control, communication, or pre/post-processing);

CNN layer tasks, representing the execution of individual CNN layers on the NPU. A CNN application is modeled as a directed acyclic graph (DAG) of dependent tasks, where each task corresponds to a single CNN layer.

Considering dependent periodic task model, each task $\tau_i \in \Gamma$ is characterized by the tuple:

$$\tau_i = \langle C_i, T_i, D_i, R_i, N_i, next_i, size_i, prec_i \rangle$$

where C_i denotes the worst-case execution time (WCET), T_i the period, D_i the relative deadline ($D_i \leq T_i$), and R_i the release time of task τ_i. The parameter N_i identifies the processing node (CPU or NPU), $next_i$ defines precedence constraints, $size_i$ denotes the resource demand (e.g., number of compute tiles required) and $prec_i$ specifies the computation precision for CNN layer tasks (e.g., INT8, FP16, FP32).

The hardware architecture is defined separately in the input file and specifies the number of CPU cores, the number of NPUs, and their execution capacities. This information is used by *npsim* to enforce resource constraints and to simulate possible contention and concurrent execution on shared computing resources.

2) Simulation Output: After simulation, *npsim* produces a schedulability report in an output file denoted `output.txt`. This report provides a comprehensive view of the system behavior, including:

- A detailed execution timeline (chronogram) of tasks on CPU and NPU resources as shown in Fig 1;
- Explicit identification of missed deadlines, if any;
- Resource utilization metrics, such as CPU and NPU usage rates;

These outputs allow system designers to assess the feasibility of a given configuration and evaluate the impact of scheduling and mapping decisions on temporal correctness.

B. Global Simulation Workflow

As illustrated in Fig. 2, the simulation workflow is organized into a sequence of well-defined stages:

1- Analysis model construction. The simulator parses the input file and constructs an internal representation of the task set. Periodic task instances (jobs) are generated over the simulation interval, and their release times and deadlines are computed.

2- Scheduling and Simulation. As shown in Fig 3, at each simulation tick, npsim updates job releases, evaluates

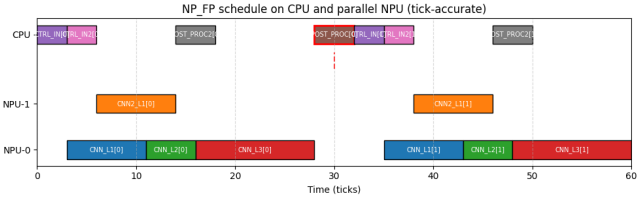

Fig. 1. Tick-accurate execution chronogram of two concurrent CNNs scheduled on a CPU–NPU architecture using NP-FP scheduling. The timeline highlights the overlapping execution of CNN layers on the NPU and control tasks on the CPU. A deadline miss is observed for one CNN, indicated by the delayed completion beyond its absolute deadline.

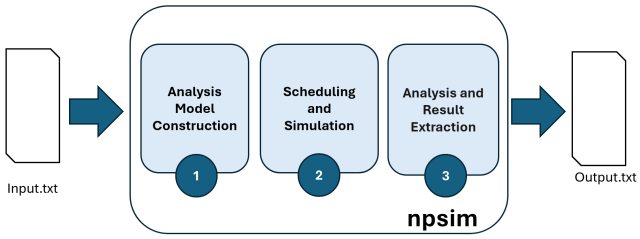

Fig. 2. Overall workflow of the *npsim* simulator. From a textual system description (`input.txt`), the simulator constructs the analysis model (1), performs tick-accurate scheduling and execution simulation (2), and extracts schedulability and timing analysis results (3), which are reported in an output file (`output.txt`).

resource availability on CPU and NPU resources, applies the selected scheduling policy, and advances the execution of active jobs. The proposed simulator aims to accurately model the execution of real-time tasks on heterogeneous CPU–NPU architectures. To this end, it enforces a set of execution and scheduling rules that capture key temporal and resource-related constraints. The following rules summarize the main assumptions applied by the simulator:

- **R1 — Dependency enforcement:** A job can start execution only after all its predecessor jobs have completed.
- **R2 — Release constraint:** A job is eligible for scheduling only after its absolute release time has been reached.
- **R3 — NPU execution model:** The NPU is modeled as a shared accelerator with possible parallel execution and non-preemptive execution semantics. Multiple NPU jobs may execute concurrently as long as their cumulative resource demand does not exceed the available NPU capacity.

$$\sum_{\{\text{ Task mapped to NPU }\}} \text{Size}_i \;\leq\; \text{Capacity}_{\text{NPU}}$$

- **R4 — Scheduling policy:** When two or more jobs simultaneously request access to the same computing resource, *npsim* resolves the contention according to the selected scheduling policy. *npsim* supports several classical real-time scheduling algorithms commonly used in CPU–NPU systems, including RM, EDF and FP.

3- Analysis and result extraction. Execution traces are analyzed to compute response times, detect deadline misses, and generate visualization and reporting outputs.

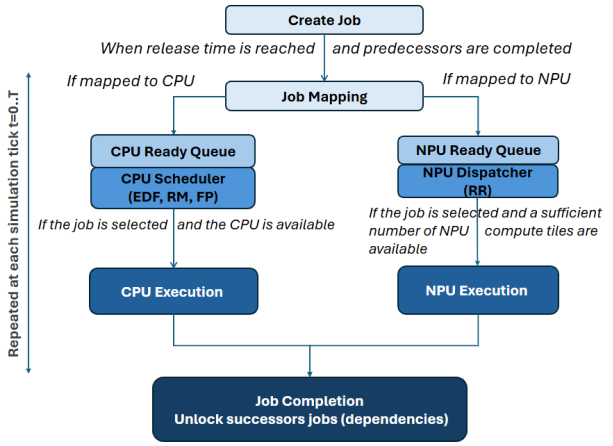

Fig. 3. Execution model of npsim. Jobs are created and become eligible once their release time is reached and all predecessor dependencies are satisfied. At each simulation tick, a mapping function dispatches jobs to either the CPU or NPU ready queue. CPU jobs are selected by a real-time scheduler (e.g., EDF, RM, or fixed-priority) and execute when the CPU is available. NPU jobs are dispatched using a non-preemptive Round-Robin policy and execute subject to NPU availability. Job completion may unlock successor jobs through dependency constraints.

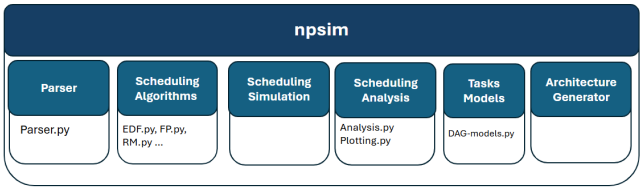

Fig. 4. Modular architecture of *npsim*.

IV. Implementation and Extensibility

As illustrated in Fig. 4, npsim is implemented in Python and follows a modular architecture to facilitate extensibility and systematic evaluation of real-time scheduling policies. The simulator is composed of independent modules. The parser (parser.py) handles input parsing and job generation. Task models are defined in the task-model directory; for example, dag-models.py provides abstractions for dependent periodic tasks modeled as directed acyclic graphs (DAGs). The architecture generator module automatically generates input configuration files, enabling the construction of synthetic and reproducible CPU–NPU platforms for experimental evaluation. Scheduling policies are implemented in the scheduling module. A tick-accurate scheduling simulation module models task execution while enforcing release times, dependencies, and resource availability. The analysis module (analysis.py) computes real-time metrics, while the visualization module (plotting.py) supports result interpretation. The overall simulation workflow is orchestrated by main.py. The source code of the proposed simulator and the experimental artifacts are publicly available at: `https://github.com/DRIDIMourad/cpu-npu-simulator`

V. Experimental Validation and Evaluation

This section presents the experimental study conducted to validate the correctness of npsim and to assess its performance and analytical capabilities. The study first evaluates the functional validity of the model and its accuracy against the NXP MCX-N platform. Then it investigates npsim's ability to capture key trade-offs between scheduling policies, numerical precision, and real-time scheduling, and finally examines its scalability.

A. Functional Validation

The first set of experiments focuses on validating the functional correctness of the simulator. In particular, we verify that the simulator accurately models the behavior of real heterogeneous CPU–NPU architectures. To this end, we validate all the properties previously identified as essential to correct system execution, including the following:

- **Dependency enforcement:** A task instance can start execution only after all its predecessor tasks have completed.
- **Release times and deadlines:** Tasks are eligible for execution only after their release times, and deadline misses are accurately detected.
- **Resource constraints:** CPU and NPU executions respect resource availability constraints, with non-preemptive execution on the NPU.
- **Heterogeneous dependencies:** Dependencies across resources (CPU→CPU, CPU→NPU, NPU→CPU, and NPU→NPU) are correctly handled.

These properties are validated through a set of unit tests and controlled execution scenarios, confirming that the simulator faithfully reproduces the intended execution semantics of heterogeneous CPU–NPU systems.

B. Accuracy Validation Against Real NPU Measurements

To evaluate the accuracy of the proposed CPU–NPU simulator, we compare the simulated response times with experimental measurements obtained on a real NPU-based platform, namely the NXP MCX-N equipped with the Neutron NPU [11]. The validation is conducted using representative CNN workloads, including SimpleNet, ResNet, Autoencoder, and YOLOv1, executed in isolation on the target hardware. All experiments are performed using FP16 precision, which represents a common trade-off between computational performance and numerical accuracy on embedded NPU platforms.

Figure 5 compares the response times estimated by npsim with the experimentally measured execution times for the considered CNNs. The results show that npsim consistently provides a safe upper bound on the response time for all evaluated networks. In particular, the relative overestimation remains moderate, with an error of approximately 25% for SimpleNet, 33% for ResNet, 15% for the Autoencoder, and 30% for YOLOv1. These results confirm that the simulator systematically guarantees conservative timing estimates while preserving a reasonable level of accuracy.

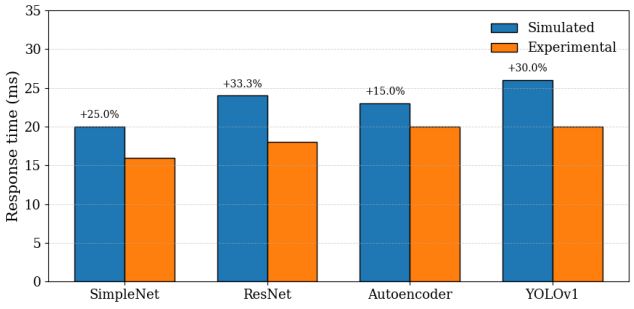

Fig. 5. Simulated vs. experimental response times for CNN workloads on the NXP MCX-N platform.

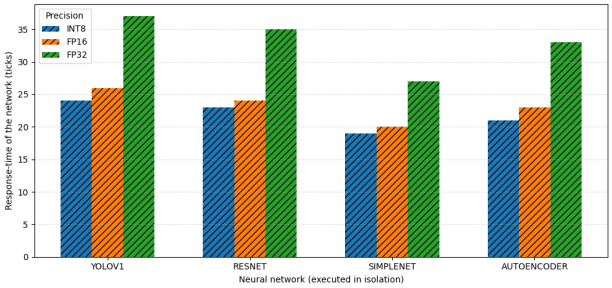

Fig. 6. Impact of numerical precision on CNN response time.

The observed gap between simulated and experimental response times reflects the inherent pessimism of the proposed system-level simulator. This pessimism can be attributed to several factors, including conservative WCET estimations at the layer level, internal hardware optimizations performed by the NPU that are not explicitly modeled, and faster-than-anticipated data transfers on the real platform. Despite these effects, npsim accurately captures the relative execution trends across CNNs of increasing complexity, demonstrating its suitability for schedulability analysis and design-space exploration of real-time CNN-based applications on CPU–NPU architectures.

C. Impact of Numerical Precision on Response Time

To evaluate the ability of the proposed simulator to capture the impact of numerical precision on execution time, we analyze several CNN workloads under different precision levels (INT8, FP16, and FP32). All networks are executed in isolation, and their response times are estimated using *npsim*.

Figure 6 clearly shows that increasing numerical precision results in higher response times for all evaluated CNNs. These results highlight the ability of the proposed simulator to model performance–accuracy trade-offs in real-time CPU–NPU systems. Figure 7 reports the schedulability of YOLO as a function of the total system utilization (CPU+NPU). The schedulability metric corresponds to the ratio of YOLO layers that meet their individual deadlines over the total number of layers. The results show that lower-precision implementations significantly improve schedulability under increasing load. The INT8 configuration remains fully schedulable up to 40% of total resource utilization, while FP16 remains

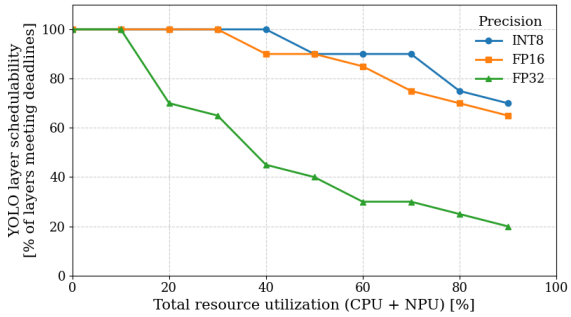

Fig. 7. YOLO layer-level schedulability as a function of total CPU+NPU utilization for different numerical precisions (INT8, FP16, FP32).

schedulable up to 30%. In contrast, the FP32 configuration rapidly loses schedulability at just 10% utilization. These results confirm that the proposed simulator accurately captures the impact of numerical precision on the timing behavior of CNN workloads. In this experiment, random background tasks are generated using UUniFast [12] to progressively increase the total resource utilization and evaluate the ability of YOLO layers to meet their real-time deadlines under varying contention levels.

D. Scalability of npsim

To evaluate the scalability of the proposed simulator, we progressively increase the number of concurrent CNN tasks and measure the total computation time of the simulation. Figure 8 reports the evolution of the simulation computation time as a function of the number of CNNs. As expected, the simulation cost increases with the size of the system. However, the growth remains controlled, enabling the simulation of realistic workloads within reasonable computation times.

Figure 8 shows that the simulation overhead grows almost linearly up to 80 tasks. A steeper increase is observed beyond 160 tasks, which is mainly due to the growing number of active jobs and the fine-grained tick-level scheduling decisions performed at each simulation step. Despite this increase, the simulator remains practical for scalability studies, completing simulations with 360 tasks in approximately 10 minutes. This overhead represents the trade-off for accurately modeling task dependencies, resource contention, and heterogeneous CPU–NPU execution semantics. These results demonstrate that the proposed simulator is scalable and suitable for analyzing systems composed of multiple concurrent CNNs.

VI. Related Work

Numerous research works have proposed simulation frameworks and analysis tools for real-time systems in order to evaluate schedulability, timing behavior, and scheduling policies. Well-known tools such as Cheddar [6] and MAST [1] have proven to be effective and are widely used for the analysis and design of real-time systems. For instance, Singhoff *et al.* [6] propose Cheddar, a scheduling analysis tool supporting classical fixed- and dynamic-priority algorithms for real-time task sets. Similarly, González Harbour *et*

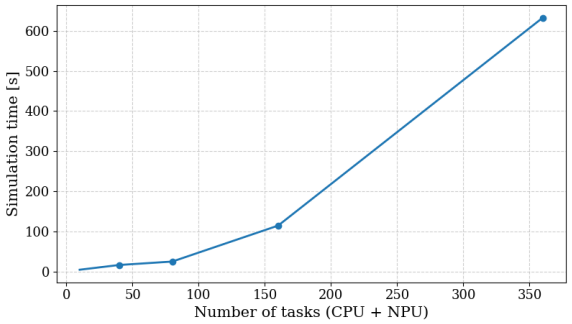

Fig. 8. Simulation time as a function of the number of tasks. Markers are shown only for representative task-set sizes to improve readability.

al. [1] introduce MAST, which enables schedulability analysis and timing verification for distributed and multiprocessor real-time systems. While these tools provide valuable support to system designers by enabling the evaluation of scheduling algorithms, task parameters, and feasibility conditions at early design stages, they primarily target mono-processor or multi-processor CPU-based architectures. As a consequence, they do not explicitly model the execution semantics and resource constraints introduced by modern AI accelerators. In particular, key characteristics of NPUs -such as possible parallel execution, non-preemptive execution semantics, data transfer overheads, and the impact of CNN numerical precision on WCRT —are generally not captured.

More recently, several works have investigated performance modeling and simulation of accelerator-based architectures for machine learning workloads, including GPUs and NPUs. A large body of literature focuses on architectural exploration using cycle-accurate simulators [2], [3] and [4]. These tools typically take as input a neural network—often described using an intermediate representation such as ONNX—together with a detailed hardware description, and produce micro-architectural performance metrics such as execution latency, number of cycles, or memory contention. Although these approaches provide high accuracy at the hardware level, they incur a significant simulation cost.

These limitations highlight the need for a simulation framework that is both easy to use and faithful to the execution semantics of real-time systems integrating CNN workloads on heterogeneous CPU–NPU architectures. In this context, we propose *npsim*, a system-level, tick-accurate simulator designed to model real-time execution semantics and to analyze schedulability while explicitly accounting for NPU architectures, task priorities, and CPU–NPU interactions.

VII. Conclusion and Future Work

Despite the increasing deployment of AI workloads in embedded systems, practical support for evaluating schedulability and timing guarantees of CNN-based applications in real-time environments remains limited. Most existing simulation and analysis frameworks do not adequately capture the execution semantics of heterogeneous CPU–NPU architectures, nor their impact on real-time behavior. In this paper, we introduced npsim, a lightweight, system-level, tick-accurate simulator dedicated to the schedulability analysis of real-time applications integrating CNN workloads on CPU–NPU platforms. By modeling CNN execution at the layer level and explicitly accounting for scheduling policies, resource contention, and numerical precision, npsim enables system designers to explore design trade-offs and reason about timing guarantees at early design stages. Future work will focus on extending the simulator to further improve its accuracy and scope. Notably, incorporating more detailed models of data transfers and communication between CPU and NPU components would enhance timing fidelity. Additional research directions include support for dynamic task mapping, mixed-criticality workloads, and more advanced NPU execution models. These extensions will contribute to making npsim a more comprehensive and practical tool for the design and validation of AI real-time embedded systems.

References

[1] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano, "MAST: Modeling and Analysis Suite for Real-Time Applications," in *Proc. 13th Euromicro Conf. on Real-Time Systems (ECRTS)*, pp. 125–134, 2001, doi: 10.1109/EMRTS.2001.934015.

[2] H. Ham *et al.*, "ONNXim: A Fast, Cycle-Level Multi-Core NPU Simulator," *IEEE Computer Architecture Letters*, vol. 23, no. 2, pp. 219–222, Jul.–Dec. 2024, doi: 10.1109/LCA.2024.3484648.

[3] H. Park, B. Kim, S. Lee, H.-J. Lee, and H. Lee, "A Cycle-Accurate Simulation Platform for NPU–DRAM Integrated Systems," in *Proc. IEEE Int. Conf. on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, pp. 1–4, 2025, doi: 10.1109/ICCE63647.2025.10929895.

[4] W. Yang, Y. Shin, O. Woo, G. Park, H. Ham, J. Kang, J. Park, and G. Kim, "PyTorchSim: A Comprehensive, Fast, and Accurate NPU Simulation Framework," in *Proc. 58th IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pp. 1363–1380, 2025, doi: 10.1145/3725843.3756045.

[5] An Zou, Yuankai Xu, Yinchen Ni, Jintao Chen, Yehan Ma, Jing Li, Christopher Gill, Xuan Zhang, and Yier Jin, "A Survey of Real-time Scheduling on Accelerator-based Heterogeneous Architecture for Time Critical Applications," *arXiv preprint arXiv:2505.11970*, 2025.

[6] M. Dridi, F. Singhoff, S. Rubini, and J.-P. Diguet, "ECTM: A network-on-chip communication model to combine task and message schedulability analysis," *Journal of Systems Architecture*, vol. 114, p. 101931, 2021, ISSN 1383-7621, doi: https://doi.org/10.1016/j.sysarc.2020.101931.

[7] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-Task Scheduling Algorithm for Preemptible Neural Processing Units" in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 220–233. doi: 10.1109/HPCA47549.2020.00027.

[8] J. Millar, Y. Huang, S. Sethi, H. Haddadi, and A. Madhavapeddy, "Benchmarking Ultra-Low-Power NPUs," *arXiv preprint arXiv:2503.22567*, 2025.

[9] T. Tan and G. Cao, "Efficient Execution of Deep Neural Networks on Mobile Devices with NPU" in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (IPSN)*, Nashville, TN, USA, 2021, pp. 283–298. doi: https://doi.org/10.1145/3412382.345827210.1145/3412382.3458272.

[10] M. Dridi, Y. Abdeddaim, and C. Daini, "Work In Progress: A New Task Model for Real-Time DNNs over GPU," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2023, pp. 337–340. doi: 10.1109/RTAS58335.2023.00035.

[11] NXP Semiconductors, " eIQ Neutron NPU for MCX N Lab Guide - Part 1 - Mobilenet - MCUXpresso SDK Builder" NXP Community, May 2023. [Online]. Available: https://community.nxp.com/t5/MCX-Microcontrollers-Knowledge/eIQ-Neutron-NPU-Lab-Guides/ta-p/1799233.

[12] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1–2, pp. 129–154, 2005.