Future Generation Computer Systems 176 (2026) 108239

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Evaluating quantile regression neural networks for optimizing real-time applications on heterogeneous platforms

Iosu Gomez [a,b,*], David Fonts [c], Sergi Vilardell [c], Unai Díaz De Cerio [a], Juan M. Rivas [b], Enrico Mezzetti [c], J. Javier Gutiérrez [b], Francisco J. Cazorla [c]

[a] Distributed and Connected Intelligence Department, IKERLAN Research Centre, Basque Research and Technology Alliance (BRTA), Arrasate-Mondragón, Spain
[b] Software Engineering and Real-Time Group, Universidad de Cantabria, Spain
[c] Barcelona Supercomputing Center (BSC), Spain

ARTICLE INFO

Keywords:
Real-time
Optimization
Heterogeneous platforms
Quantile prediction
Memory contention

ABSTRACT

Modern cyber-physical systems increasingly rely on computationally demanding applications, particularly at the edge, where Artificial Intelligence-based algorithms are deployed. To meet these demands, industry trends are shifting towards heterogeneous MultiProcessor Systems on Chip (MPSoCs), which must also satisfy strict real-time and functional safety requirements. A major challenge in such systems is memory contention, where multiple processing units compete for shared memory resources, affecting application performance and the accurate estimation of Worst-Case Execution Times (WCETs). Traditional static analysis becomes impractical as system configurations grow in complexity. This work presents the design of an analysis and optimization framework for real-time systems that re-evaluates WCET estimates based on system configurations to reflect the impact of memory contention on heterogeneous platforms. The proposed method estimates new WCETs using Quantile Regression Neural Networks (QRNNs), which infer memory contention from Event Monitor data. Experimental results reveal that QRNN models must be system-specific for accurate predictions and that memory access patterns significantly affect model generalization. Two strategies are proposed: using generic models for simplicity or task-specific models for higher accuracy. Despite some potential underestimations, QRNNs maintain a strong correlation with actual observed contention, enabling effective worst-case scenario identification. Furthermore, a comparative analysis highlights the superior scalability of the estimation-based approach over empirical measurements, especially in large system optimization processes where performance can be easily enhanced by at least two orders of magnitude, making it a practical solution for real-time system design and analysis.

1. Introduction

1.1. Background

Modern cyber-physical systems must support increasing computational demands. For instance, the European Cloud, Edge and IoT Continuum [1] initiative offers a new perspective on how to approach the development of new technologies that seek to reduce the communications burden by processing data closer to where it is generated. In particular, at the edge side, the use of computational-intensive algorithms, such as those used in Artificial Intelligence applications or in advanced perception systems, demands the usage of advanced hardware platforms capable of satisfying those computing requirements. The trend in the industry is advancing toward heterogeneous platforms that integrate multiple processors and specialized accelerators within a single MPSoC (MultiProcessor System on Chip). These platforms are not limited to regular cores, but can also incorporate specialized processors that operate at different frequencies and contain more complex memory hierarchies. In addition, they can include dedicated hardware accelerators, which allow reducing the execution times of specific jobs. These characteristics make MPSoCs great candidates for such applications. At the same time, these systems must not only guarantee a correct functionality but also comply with functional safety and real-time requirements. This research domain is of great industrial interest, as reflected in different industrial challenges presented at international conferences. A notable example is the Arm industrial challenge presented at the ECRTS conference [2], which addresses an industrial use case based on a smart mobility application.

Modern MPSoC platforms present several challenges for ensuring time predictability. A fundamental challenge lies in the scheduling of

* Corresponding author at: IKERLAN Research Centre, Spain.
E-mail address: iosu.gomez@ikerlan.es (I. Gomez).

https://doi.org/10.1016/j.future.2025.108239
Received 2 July 2025; Received in revised form 30 October 2025; Accepted 1 November 2025
Available online 5 November 2025

these systems, as tasks running on different processing resources must be synchronized and efficiently assigned [3,4]. However, one of the major challenges is modeling memory contention [5,6]. Contention arises when multiple processors or dedicated accelerators compete for the usage of shared memory resources such as caches or global memory. The intensive memory usage by multiple computing resources may increase cache misses and saturate the access to the memory device, directly impacting the execution times of the applications and leading them to not meet timing constraints. Furthermore, static timing analysis and system optimization can also be adversely affected by memory contention, as the impact on the Worst-Case Execution Times (WCETs) depends on the system configuration and the underlying memory subsystem. WCETs are needed by common techniques used in real-time optimization processes. Consequently, for each system configuration generated by an optimization process, the analysis process shall re-evaluate the WCET estimates for the specific configuration, which introduces additional challenges. One possible approach to address this challenge involves empirically producing WCET estimates for every configuration by deploying tests on the real platform. However, this option can be infeasible due to the high execution times required to perform all the computations.

A more feasible alternative is relying on estimation techniques capable of analyzing the memory subsystem's behavior and providing reliable predictions of the memory contention without the need for exhaustive testing work on real platforms [7]. Nevertheless, the challenge of these analytical approaches is their high complexity in multicore settings, as the contention scenarios can grow exponentially due to the complex inter-dependencies that may arise. As a result, the analysis could become computationally intractable, or it may be highly pessimistic. Although estimation-based approaches, such as probabilistic WCETs (pWCETs), do not define deterministic bounds, they have been widely studied as a viable alternative for modern architectures, where traditional methods are unfeasible or excessively pessimistic [8,9]. In the context of safety-critical systems, certification of estimation-based approaches may be challenging due to possible underestimations. In this case, additional verification of the estimated values would be required. However, they can be directly integrated into mixed-criticality systems where less critical tasks are allowed to have a certain failure rate [10].

1.2. Contributions and paper organization

In this study, we propose exploring a new avenue to perform WCET analysis that, instead of relying on complex formulations, leverages the use of contention estimation techniques such as QRNN (Quantile Regression Neural Network) [11,12] to perform the estimation of the WCETs more efficiently. The QRNN is trained to infer the memory contention that a task may experience based on the Event Monitors (EMs) of all the tasks executed across the multiple processors. By using EMs, the QRNN model is provided with the necessary information to model the performance of the memory subsystem's behavior to make accurate predictions.

The main contribution of this paper is the evaluation of the applicability of QRNN-based techniques for their integration into a wider framework for the analysis and optimization of multicore real-time systems. In contrast to traditional methodologies for single processors [13], where WCET estimates are considered fixed and known in advance, the proposed framework enables dynamic re-evaluation of the WCET estimates based on the system's configuration in order to reflect the impact of memory contention. This approach aims to open new avenues for the design and analysis of modern real-time systems deployed on MPSoCs, where high computational and intensive memory usage applications introduce new challenges that demand reconsidering how these systems are studied.

This manuscript is organized as follows. Section 2 introduces related work on WCETs analysis under memory contention. Section 3 provides an overview of QRNNs. In Section 4, the analysis and optimization framework for heterogeneous platforms is presented. Section 5 depicts how the QRNN should be integrated into the framework to produce re-evaluated WCET estimates. Section 6 describes all the given steps in order to apply the QRNN into a heterogeneous platform, from how data was captured to describe the training process. In Section 7, we evaluate the applicability of QRNN-based techniques for the proposed framework based on experimental results. Finally, Section 8 shows the main conclusions and future work.

2. Related work

Memory contention is a critical factor in real-time systems as it can increase execution time and hence WCET estimates, which may lead to deadline misses. Several works [6,14,15] have developed analytical techniques to obtain upper-bound estimates of the WCETs under contention scenarios in multicore systems. In [16], the authors extend these techniques to heterogeneous platforms that integrate multiple types of accelerators (in addition to CPUs) in the same MPSoC, although their work is limited to global memory contention.

Analytical WCET estimation methods have also been leveraged to improve response time analysis techniques for multicore systems, where contention is a critical aspect to be considered. Based on WCET analysis techniques proposed in [14], the work [17] presents a response time analysis that considers memory contention. Additionally, other studies [18–21] have contributed to the development of methods for predicting upper-bounds on memory contention and integrating them into response time analysis.

Traditional WCET analysis techniques focus on deterministic upper-bound estimations with probabilistic approaches gaining importance due to their ability to provide more flexible solutions in systems with higher complexity, where several inter-dependencies may result in unfeasible analytical calculations or very pessimistic estimations. Probabilistic WCET (pWCET) analysis, such as the ones explored in [8] and [9], provides a statistical upper bound on execution times, considering the inherent variability of modern architectures. In our case, the proposed framework is aligned more closely with probabilistic WCET methodologies, where a statistical upper bound is given to the variations introduced by memory contention.

The complexity of systems grows exponentially in modern MPSoCs by the increasing number of cores. Consequently, as modeling all these inter-dependencies is becoming a major challenge, WCET estimation techniques are relying more and more on Machine Learning (ML) approaches. Works such as [22] and [23] proposed ML-based techniques to analyze WCETs in these complex scenarios. Other studies [24,25] have incorporated memory contention prediction capabilities into the neural networks to effectively estimate WCETs. In [12], a neural network based on quantile regression is proposed to predict contention with a controlled overestimation, which allows for increased reliability of the predictions. Similarly, [26] proposes a machine learning-based technique for estimating WCET, although it focuses on single-core and does not consider contention effects. Finally, [27] proposes the use of neural networks to predict contention generated by shared resources on GPUs. In our work, as it is focused on predicting the multicore contention, we propose the integration of the technique developed in [12], QRNN, into a proposed analysis and optimization framework that considers how much WCETs increase due to memory contention, evaluating the reliability of predictions across different system configurations. To the best of our knowledge, few works operate in the same manner as QRNN. Therefore, the similarities with previous works end in the use of NN and WCET estimation because: (1) they do not aim at estimating WCET under contention (multicore) scenarios, (2) they aim at estimating the WCET of a Basic Block (BB), we aim at the whole workload, and (3) they use the semantics of the assembly code of the BB as variables, while we use Hardware Event Monitors. Therefore, QRNN seeks to predict a much more complex timing profile, given the variability present in multicore.

In the context of optimization, [28] proposes a methodology to mitigate the effect of memory contention by mapping tasks to processors, using the memory contention prediction methodology developed in [6]. In [29], the authors present an offline/online multistage optimization framework for multicore platforms that combines mapping, scheduling, and buffer allocation. The work in [30] presents a contention-aware analysis method along with task allocator mechanisms to improve schedulability by reducing memory contention. Unlike these optimization techniques, our proposed framework is not focused on optimizing response times by mitigating memory contention. Instead, we propose a framework with an open scheme that allows the integration of different tools to optimize any desired parameter by changing the system configuration for which the memory contention can be estimated for any tested configuration (e.g., applying well-established optimization algorithms such as HOSPA [31] or MILP [32]). The proposed approach enables exploring configurations where, even though memory contention can be higher, optimization objectives might still be achieved through other system parameters. Moreover, it can be integrated into other frameworks that leverage the use of supercomputers to evaluate optimization techniques themselves [33].

3. Quantile regression neural network to predict WCETs

The objective of this section is to describe the QRNN, a modeling framework that supports the optimization procedure described in Section 4. Specifically, we aim to estimate the WCET of a task τ, running on one core of an MPSoC, while the remaining cores are simultaneously executing other tasks. In such heterogeneous systems, tasks share hardware resources such as caches, memory controllers, and interconnects. As a result, when multiple tasks run simultaneously, they compete for these shared resources, leading to increased execution times compared to when the same task runs in isolation. To formalize this scenario, we define a workload $w = \{\tau_1, \ldots, \tau_K\}$, as the set of K tasks running simultaneously, one per core, in an MPSoC with K cores. For each workload w, our goal is to use QRNN to estimate the WCET of a given Task Under Analysis (TUA), considering the contention introduced by the co-running tasks.

3.1. Introduction

As we mentioned, tasks in a MPSoC compete for hardware shared resources, leading to contention that affects tasks Execution Time (ET) of the TUA, τ. Our goal is to use a set of representative EMs extracted from executions in isolation ($ET_0(\tau)$) to derive a Time Budget in Multicore ($TB_w(\tau)$), when a TUA run as part of a specific workload w, with other tasks, with a contention factor $P\Delta_w$ and an inherent ET variability of the MPSoC, which we denote as ϵ:

$$TB_w(\tau) = ET_0(\tau) \times P\Delta_w + \epsilon, \tag{1}$$

We define our contention model for $P\Delta_w$, as a function f of EM values , aiming to estimate a contention factor for the given TUA within w.

$$\widehat{P\Delta_w} = f(EM_w; \Lambda), \tag{2}$$

where Λ represents the model parameters, and the hat notation indicates that $\widehat{P\Delta_w}$ is an estimation of the true contention factor $P\Delta_w$.

However, since the true contention factor $P\Delta_w$ is unknown, we consider the observed contention, defined as the ratio between the measured execution time and the nominal execution time:

$$O\Delta_w = \frac{TB_w(\tau)}{ET_0(\tau)}. \tag{3}$$

This observed value $O\Delta_w$ serves as a practical estimate of the contention experienced by the TUA within workload w.

QRNN optimizes the quantile regression loss function, which is designed to approximate any desired conditional quantile of $TB_w(\tau)$. This flexibility enables users to select a high quantile that best suits their specific needs. QRNN provides an efficient and effective method for evaluating system configurations. By offering conservative yet accurate predictions of contention impact, they facilitate informed decision-making in resource allocation and scheduling, ultimately improving system performance in multicore environments.

Table 1
Example of EM values for two tasks.

	CPU Cycles	Memory Accesses	…	L2 Refills
Task 1	100	25	…	1
Task 2	90	30	…	3

3.2. Representative EMs

When predicting a $TB_w(\tau)$, we only know which tasks are in the workload w and their EM data. The core idea of the QRNN is to leverage EM data from tasks executed in isolation on the same platform (as presented in Table 1), which is far more practical than measuring EMs for every possible workload and core.

The challenge in this process is that EMs cannot be fully captured from a single execution due to physical constraints, with the Performance Monitoring Units (PMUs), which will be detailed in Section 6. Instead, they can only be retrieved in groups of 4–8 EMs, all sampled simultaneously during the same run [12], limiting the ability to obtain a complete set of EMs from a single run.

Previous works [34] have shown that the hardware and software state change across runs of the same workload, creating some noise in the EMs read. In order to properly merge EMs collected in different runs, we employ the Merging Hardware Event Monitors (HRM) algorithm, which aligns different blocks by leveraging the order statistics of execution time as an anchor. This alignment ensures that the reconstructed EM vector maintains plausibility, correlation, and structural consistency across multiple executions.

With this approach, each task τ_i is assigned a single representative set $EM(\tau_i)$.

For instance, the corresponding EM vector of Task 1 would be $EM(\tau_1) = [100, 25, \ldots, 1]$, and the complete workload representation can be expressed as $EM_w = concat(\mathrm{EM}(\tau_1), \ldots, \mathrm{EM}(\tau_k))$.

Given a workload for which we aim to predict the contention of the TUA, we replace each τ in the vector w with its corresponding EM vector. This transformed representation, that we call EM_w is then fed into the QRNN model, which predicts workload contention by analyzing the inferred interactions between tasks.

3.3. Neural network (NN) models

Neural networks (NNs) capture complex, non-linear relationships by processing inputs through weighted transformations and activation functions. In our approach, the NN takes a vector of representative EMs, EM_w, summarizing isolated task executions, and estimates the contention factor for a workload w:

$$\hat{y}_w = \phi(EM_w; \Lambda) = \widehat{P\Delta_w}, \tag{4}$$

where Λ are the model parameters. These parameters are optimized by minimizing a loss function $\mathcal{L}(\widehat{P\Delta_w}, O\Delta_w)$, which quantifies the prediction error. By reducing this error, the NN learns to capture workload interactions and generalize across execution scenarios.

The choice of loss function determines which aspect of contention is optimized. In particular, for WCET estimation, instead of using the mean, the QRNN uses a loss to estimate a target quantile γ.

3.4. Quantile regression

Quantile Regression (QR) extends traditional regression techniques by estimating conditional quantiles instead of mean values. This is

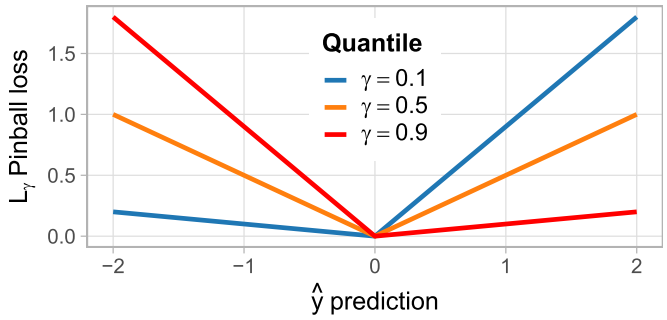


Fig. 1. QR loss function shape centered at zero. The prediction $\hat{y}$ in the horizontal axis and the vertical axis corresponds to the loss value of the example in Section 3.4.

particularly useful in contention modeling, where extreme cases (e.g., high contention scenarios) must be accounted for. The QR loss function, known as pinball loss, penalizes the residuals $u = y - \hat{y}$ as follows:

$$\rho_\gamma(u) = \begin{cases} \gamma \cdot u, & \text{if } u \geq 0, \\ (\gamma - 1) \cdot u, & \text{if } u < 0. \end{cases} \tag{5}$$

The pinball loss function introduces asymmetric penalties based on the parameter γ, which determines the conditional quantile being estimated. When $\gamma = 0.5$, it symmetrically penalizes over- and underestimations, reducing to the absolute error and estimating the median. As γ deviates from 0.5, the loss function becomes asymmetric, assigning greater penalties to underestimations when $\gamma > 0.5$ and to overestimations when $\gamma < 0.5$.

We can visualize it in Fig. 1 with an example where the pinball loss (Eq. 5) is evaluated under three different quantiles and the real value is $y = 0$. For instance, when $\gamma = 0.9$, an error of $|u|$, is penalized 9 times more severely when it is an underprediction; this asymmetric loss drives the model to have 9 times more overestimations than underestimations, effectively estimating the 90th percentile.

3.5. Quantile regression neural network (QRNN) models

QRNNs integrate the strengths of quantile regression and neural networks, enabling the estimation of specific execution time quantiles while capturing complex interactions in workload contention.

QRNN uses 4 fully connected hidden layers with 300 units each and ReLU activations to optimize the pinball loss function (Eq. 5) using the Adam optimizer. This setup allows the model to learn contention predictions tailored to a chosen quantile γ. The selection of γ determines the focus of the model: lower quantiles prioritize underestimation control, while higher quantiles emphasize worst-case contention scenarios. Since γ is a fixed parameter, a separate model can be trained for each desired quantile.

In our study, we aim to reduce underestimations, i.e., cases where the predicted contention $\widehat{P\Delta_w}$ falls below the $O\Delta_w$. While the model tries to be as accurate as possible, overestimations are preferred over underestimations in systems with real-time constraints.

To mitigate this risk, we train QRNN on the 90th quantile ($\gamma = 0.90$) of $O\Delta_w$, ensuring that approximately 90 % of the predictions are overestimations. Note that, this selection of $\gamma = 0.9$ could be modified, allowing us have flexibility on the requirements of the prediction.

4. Optimization framework for heterogeneous platforms

The optimization of real-time systems is a process that aims to find an optimal or optimized system configuration which involves, for example, ensuring that tasks meet their timing requirements or just improving specific metrics of the system. As part of these processes, it is common practice to perform a response-time analysis on a given system configuration to determine the Worst-Case Response Times (WCRTs) of all the tasks within the system. This analysis provides the necessary information that optimization algorithms can use to assess the quality of intermediate solutions, and to guide the optimization process towards better solutions. For instance, the work [35] proposes a methodology to model and analyze real-time applications that may include GPUs. This methodology uses different variants of the offset-based response-time analysis techniques to validate the real-time requirements over hierarchically scheduled time-partitioned systems, i.e., time partitions can be defined at the primary scheduler as a set of partition windows that are cyclically executed, and preemptive fixed priorities are used by a secondary scheduler within each partition. Optimization may be performed to assign priorities, map tasks to partitions, map partitions to processors or to configure partition window sizes.

Response time analysis techniques require that WCET estimates have been determined in advance for every piece of code. However, the use of heterogeneous platforms with multiple processors may lead to memory contention [36], which has a direct impact on the WCETs. This dependency makes it difficult to obtain precise measurements of WCETs, or it may lead to very high values of WCET estimations by considering overly pessimistic worst-case situations.

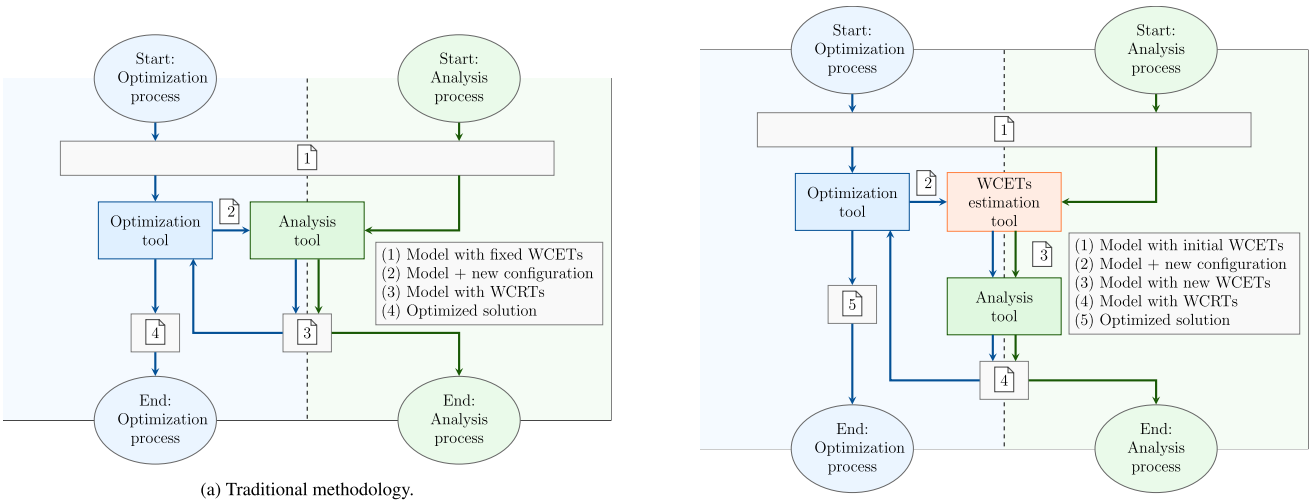
The hierarchical optimization of real-time applications involves several inter-dependent sub-problems related to the definition of the time partitions and mapping of partitions, tasks and priorities. As a result, optimizing such systems represents a complex challenge, as each configuration may produce different contention conditions, which require continuous reevaluation of WCETs.

In the traditional methodology for implementing analysis and optimization processes of real-time systems, the WCET estimates of the tasks that compose the system are measured or analytically estimated through worst-case assumptions. Once the WCET estimates have been obtained, they are assumed constant in all further steps of the overall optimization and analysis processes. This traditional methodology is represented in Fig. 2a. In this optimization process, from an initial input model (1), including timing (particularly WCET estimates) and platform properties, a new configuration may be proposed by an optimization tool (e.g., partitions, mappings, priorities, etc.). This configuration (2) is analyzed by using any available response-time analysis technique. As a result, WCRTs are obtained (3). Based on the obtained values, the optimization tool decides whether a valid solution has been obtained (4) or a new configuration should be tried (2), repeating the cycle.

However, when the proposal of a new configuration may lead to variations in the memory contention profile, the WCET estimates can no longer be considered fixed values. Therefore, the proposed methodology takes into account these inter-dependencies, by adding an additional process that evaluates the WCET estimates for each proposed configuration, so both the optimization and analysis processes operate with appropriate WCET values for each configuration.

This new scheme, represented in Fig. 2b, is suitable for modern MPSoC platforms, and consists of the following steps. First, the input model of the real-time system (1) is provided, including an initial configuration of the system. This initial model may contain WCET estimations, but these are not necessary because the optimization cycle incorporates a WCET estimation step. Then, the optimization tool proposes a new configuration (2), which is then validated by first estimating the updated WCETs (3) and finally performing a schedulability analysis which yields updated WCRTs (4). As in the traditional methodology, if the current configuration is determined to comply with the optimization objectives, the algorithm returns this configuration as the final solution (5). Otherwise, the optimization process is repeated.

In order to generate the new WCET estimates for each configuration in the proposed methodology, two approaches can be applied. One approach involves deploying test code on the real platform to empirically measure the WCETs of the tasks under the specific system configuration. This test code would measure the WCETs under the contention



(a) Traditional methodology.

(b) Proposed methodology for MPSoC hardware.

Fig. 2. Traditional vs proposed methodology for the analysis and optimization process of real-time systems.

conditions caused by the input configuration. The major issue of this approach is the time required to measure the WCETs; to obtain reliable measurements, it is necessary to repeat the experiments thousands of times, which is highly time-consuming. Additionally, these measurements should be performed at each iteration of the optimization process, further increasing the time required to perform the whole process. Another alternative to implementing the proposed scheme in a more efficient way is to use a WCET analytical technique [6,14,15]. These techniques can obtain the WCETs more efficiently by not relying on deployments on real platforms.

However, analytical approaches face significant challenges due to the complexity of modern MPSoC platforms. As mentioned earlier, contention scenarios can grow exponentially with the increase of task interdependencies, leading to computationally expensive analysis or highly pessimistic assumptions. As an alternative, emerging estimation techniques such as QRNN are becoming a more viable option, offering an efficient solution for WCET analysis without relying on complex analytical methods.

Unlike previous frameworks such as the one presented in [19], where memory contention was analytically calculated in order to perform response-time analysis, our scheme allows a modular integration of existing and well-established analysis and optimization tools like MAST [37]. In their original form, these techniques do not support the re-evaluation of WCETs or the analysis of memory contention. While assuming the worst-case scenario of contention in each task is possible, this approach would be highly pessimistic. Our framework allows for more realistic analysis and optimization for MPSoC platforms. By incorporating WCET estimation techniques, we overcome this limitation of traditional analytical methods, whose complexity grows exponentially in modern MPSoC systems. Therefore, our framework maintains compatibility with well-established analysis and optimization techniques while offering a scalable and efficient approach for dynamic re-evaluation of WCET estimates.

As mentioned before, each step proposed in the optimization process proposes a new system configuration, which may include task to core allocations, assigning tasks to partitions, or other scheduling parameters. This decision directly affects the memory contention, as different tasks may compete for shared resources in each configuration. Consequently, the system's WCET estimates may vary, which modifies the response times. If the resulting configuration does not meet the optimization objectives, the framework generates a new configuration, which again requires dynamic re-evaluation of the WCET estimates. This process ensures that the impact of optimization on memory contention is reflected and that the system is analyzed realistically under each proposed configuration.

5. Integrating the QRNN into the optimization framework

As presented in Section 3, the QRNN needs an input vector containing the EMs for each workload to generate predicted contention factors for every TUA. In contrast, in the proposed analysis and optimization framework presented in Fig. 2b, the WCET estimation tool would be fed with the system's model, including timing parameters and a configuration. This tool then produces an updated model containing the adjusted WCET estimates taking into account the new system configuration. This section outlines how the QRNN should be integrated into the WCET estimation tool to generate this output model. For this purpose, the input model must contain:

- The real-time system description, including timing and platform properties. The contained WCETs must be measurements or estimates of tasks in isolation. These initial values will later be adjusted by applying the predicted contention factors.
- A system configuration, which is typically computed by an optimization tool. This configuration may include partition settings, task mappings, priorities, etc.
- A vector of EMs measured in isolation for each task that will be used by the QRNN to predict the contention factors.

Fig. 3 depicts the scheme needed by the WCET estimation tool to obtain the output model with new WCETs given the data described above. The proposed scheme consists of the following steps:

1. Contention analyzer: The contention analyzer receives as input a model with all the information detailed above. This model may be provided by the optimization or analysis process (1,2). Its main function is to identify, for each TUA, the workload combinations that may cause contention during its execution. These workloads depend directly on the system configuration, which includes decisions such as partitioning (if applicable) or task-to-core assignment. As output, the contention analyzer would generate a file containing the EMs vectors for each identified workload (a).
2. QRNN: The QRNN takes as input the EMs vectors for each workload (a), and generates a file with the predicted contention factor for each workload (b).
3. WCET calculator: The WCET calculator takes the initial system model (1,2) and the predicted contention factors for each workload

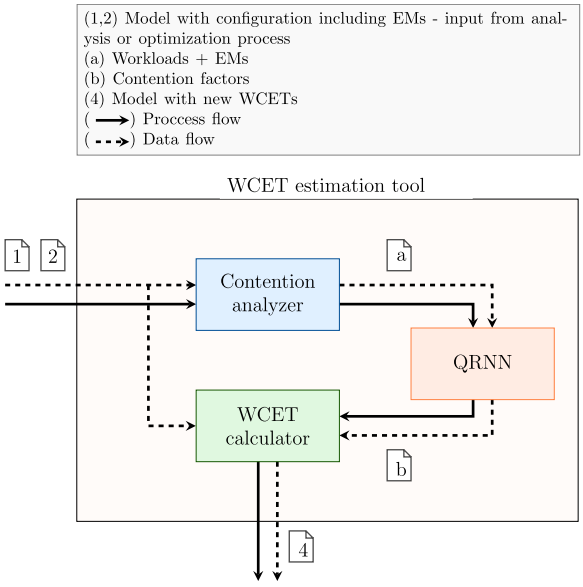


Fig. 3. Possible integration of QRNN into the WCET estimation tool.

(b). This step adjusts the initial WCETs by applying the predicted contention factors. To determine the appropriate contention factor to apply, the worst-case scenario is assumed, selecting the highest possible predicted contention for each TUA. The final output (4) is an updated system model containing the previous information along with the adjusted WCET values.

As an illustrative example showing how the process in Fig. 3 works, consider a simple configuration of three tasks $\{\tau_1, \tau_2, \tau_3\}$ to be mapped into 2 cores. Each task has a WCET estimated in isolation, $ET_0(\tau)$, and the associated EM vector obtained by profiling, which includes the execution time in isolation. In this scenario, the contention analyzer identifies the potential interfering workloads from configuration (1). In the case of τ_1, $w_{1,1} = \{\tau_1, \tau_2\}$ and $w_{1,2} = \{\tau_1, \tau_3\}$, excluding core permutations for simplicity. For each of these combinations, the corresponding EM vectors (a) are provided by extracting the information from EMs in (2). As an example, the EM vector for $w_{1,1}$ is composed by a concatenation of EMs of tasks τ_1 and τ_2, $w_{1,1} = concat(EM(\tau_1), EM(\tau_2))$. Then, the QRNN predicts the contention factor for each workload (b), denoted as $\widehat{P\Delta}_w$. For the case of τ_1 contention factors $\widehat{P\Delta}_{1,1}$ and $\widehat{P\Delta}_{1,2}$ are obtained. Finally, the WCET calculator updates the execution time for each task by multiplying the worst contention factor by the execution time in isolation. The conservative estimate of τ_1 would be obtained as follows: $ET(\tau_1) = max(\widehat{P\Delta}_{1,1}, \widehat{P\Delta}_{1,2}) \times ET_0(\tau_1)$. For each task, the same steps are followed. As previously mentioned, the updated execution time estimates are included in (4).

The computational complexity of this scheme will be dependent on the specific analysis and optimization algorithms that can be used. In any case, the demonstrated scalability in Section 7.3.3 will further benefit the most complex optimization algorithms.

6. Workloads, data acquisition and training

6.1. Testing platform

As a test platform, we used an NVIDIA Jetson AGX Xavier [38] device. This heterogeneous platform features the Carmel CPU Complex and a 512-core Volta GPU, along with other dedicated accelerators. We focus on the CPU complex (see Fig. 4) that is composed of four clusters containing two Carmel cores each. These cores are NVIDIA's proprietary design based on the ARMv8 architecture. Each core includes a 128KB instruction and a 64KB data L1 cache, while the two cores within each

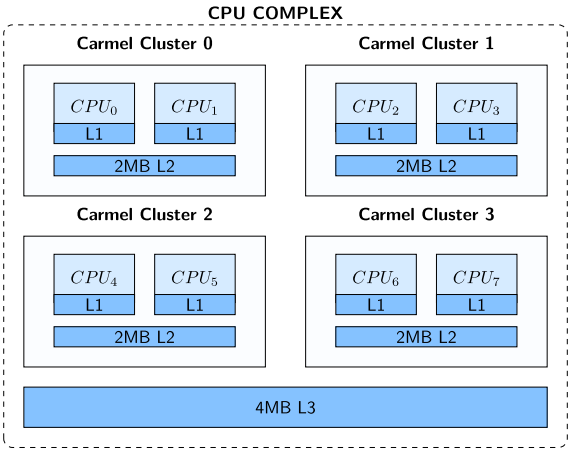


Fig. 4. Carmel CPU complex.

cluster share a 2MB L2 cache. The four clusters share the 4MB L3 victim cache included in the System Coherency Fabric (SCF), which is responsible for connecting the Carmel Cores with the memory interface.

6.2. Core and uncore performance monitoring units

As the Carmel cores are based on the ARMv8, they also provide access to the Performance Monitoring Units (PMUs) designed by Arm. These PMUs allow monitoring core-specific hardware events or event monitors (EMs) such as CPU cycles or memory reads/loads. Due to design limitations, certain EMs in Carmel cores remain inaccessible through conventional PMUs. Specifically, those related to shared resources like L2 cache and global memory that occur outside the core (e.g., global memory accesses are requested by the SCF).

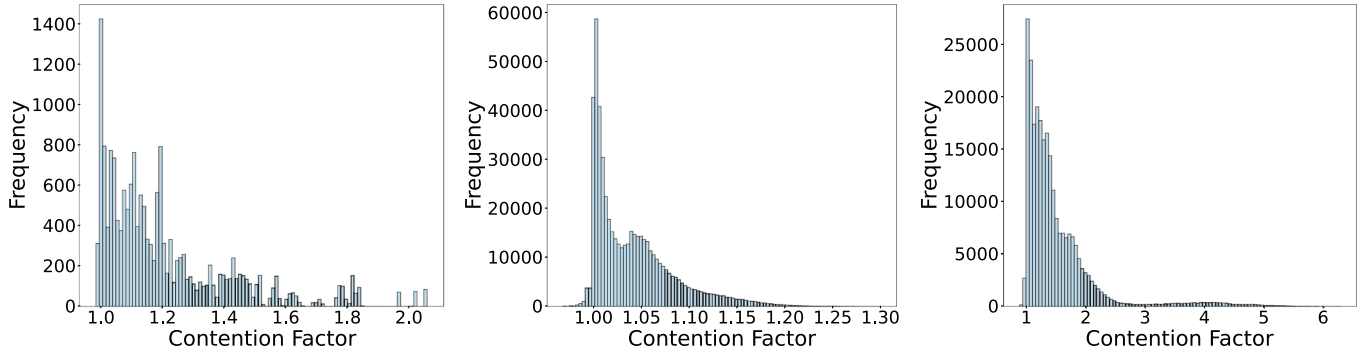
To address this limitation, the Carmel Complex includes the Uncore Performance Monitoring Units (U-PMUs). These units provide access to those EMs that occur outside the cores. However, the U-PMUs do not allow identification of the association between a specific EM and the core to which this EM belongs. This fact does not represent a problem in our case because, as will be detailed later in this paper, the EMs are monitored with tasks running in isolation, with no other task executing on the rest of the cores. In this way, we can ensure that the monitored EMs originate from the task being analyzed.

A common practice to access the PMUs and capture EMs is to use the *perf_events* library [39], which serves as a standard interface to performance monitoring in Linux systems. However, after testing it on the Jetson AGX Xavier platform, we observed that this library does not provide full access to all the available EMs on this platform. In particular, it does not support EMs related to L2 cache and memory access, which are part of the uncore EMs. To gain access to all available metrics (both core and uncore), a custom kernel module was developed. This module interfaces directly with the PMU and U-PMU registers, allowing user-space applications to configure and read these registers. In this way, we obtained access to all the available EMs, a total of 36 (27 core and 9 uncore). These events are summarized in A.

In summary, the available EMs offer broad coverage of different aspects of the memory hierarchy, from cache to memory bus usage patterns. This variety allows the QRNN to learn how different parameters may affect the application's execution times. Consequently, the prediction model can recognize patterns associated with different forms of memory contention, from cache misses caused by other cores overwriting data in a shared cache to main memory access saturation.

6.3. Workloads

To achieve a comprehensive analysis, it is essential to consider tasks with different memory usage patterns. To address this, a set of



(a) Single contender setup. (b) Three contender setup. (c) Full contender setup.

Fig. 5. Observed contention factor distribution across different setups: (a) Single contender, (b) Three contender, and (c) Full contender.

custom benchmarks has been used that emulate basic operations commonly used in NN applications or in advanced perception systems. These benchmarks are categorized as follows:

- Memory access: Tasks performing random memory accesses, emulating data load, and storing operations typically used in regular applications.
- Matrix operations: Multiple matrix operations that can be found in NNs and machine learning algorithms. These include matrix addition, multiplication, transposition, and a combination of transposition with multiplication.
- Vector operations: Multiple vector operations that can be found in a variety of applications like signal processing.
- Activation operations: Activation functions that can be found in NNs or signal processing, such as rectifier and scaling functions.
- LIDAR processing: LIDAR point-cloud processing operations, which involve downsampling and filtering algorithms.
- Railway application based on a genetic algorithm, used in [35].

Some of the described benchmarks may have different variants depending on their data footprint. In the *dl1* variant, the data used by the benchmark fits in the L1 data cache, allowing the CPU to access it with minimal latency. In the *l2* variant, the data does not fit in the L1 cache, but it does fit in the L2. Finally, in the *mem* variant, the data length is too large to fit in any cache. Thus, the CPU is forced to access the main memory. For benchmarks that do not have a specified variant, the *mem* variant is assumed by default.

No variant is defined for the L3 cache, as in this platform it acts as a victim cache, its content depends on cache misses from the upper memory levels rather than direct data placement. This makes a total of 24 custom benchmarks, which are further described in B.

6.4. Data acquisition

As we previously discussed, we aim to estimate the contention factor, $\widehat{P\Delta_w}$, from the representative EMs of each benchmark in workload w. To obtain the data set needed to train the QRNN, a sequence of measurements have been performed on the testing platform. As a first step, we measured the EMs for each benchmark by executing them in isolation, to build the representative EMs. The next step involves executing benchmarks in parallel distributed across different cores to measure the impact of contention $TB_w(\tau)$. Then, the contention factor of each workload is calculated $O\Delta_w$.

In order to perform the measurements in isolation, since the number of EMs to be captured exceeds the number of Performance Monitoring Counters (PMCs) in the PMUs and U-PMU, multiple runs are required. To handle this, following [34], we selected the execution time (ET) as a fixed reference (or anchor) EM included in each read, obtained by tracing the *cpu cycles* event. This anchor provides a common reference that allows us to later align the EMs of different reads. In total, 5 independent reads are required to cover the 35 EMs (we discarded the INST_RETIRED event due to inconsistent behavior). In order to obtain a comprehensive view of the variability of the results, these 5 reads were repeated 100 times for each benchmark. To eliminate anomalies, we filtered out periodic peaks detected in the execution traces, attributed to overheads caused by de OS. Any execution that was removed also had its associated EMs removed to maintain consistency in the dataset.

To obtain the impact of memory, the TUA is monitored by measuring its execution time while contention is generated by running other benchmarks on the available cores. Three different setups were considered:

- Single contender setup, where contention is generated within the same cluster as the TUA.
- Three contender setup, where contention is generated by a single core from each cluster outside the TUA's cluster.
- Full contention setup, where contention is generated by every available core in the CPU complex.

Fig. 5 shows the distribution of the observed contention factors obtained for the different benchmarks in each setup:

- Single contender setup (Fig. 5a): A contention factor up to 2.06 is observed as the contender shares the L2 with the TUA.
- Three contender setup (Fig. 5b): Despite having more tasks in contention, tasks do not compete for the L2 cache, which resulted in a reduction of contention to 1.3.
- Full contention setup (Fig. 5c): Contention occurs at all memory levels, increasing memory access requests, and amplifying the effect of the memory contention up to 6.28.

It is important to note that contention factors below one appear because median values of execution times in isolation were used as a reference to calculate the factors. Moreover, in the same way as in the experiments in isolation, the filtering process was also applied in order to minimize the effect of the overheads caused by the OS.

Finally, the contention factor of each TUA under a workload w is obtained by dividing the measured execution time in contention by the execution time in isolation.

6.5. Preprocessing

The objective of the preprocessing is to construct a dataset from the data acquired in the previous section. To this end, the following steps were followed: (1) identification of representative executions, (2) the construction of the dataset from the contention data, (3) the balancing of the dataset, and (4) the addition of the representative EMs to the dataset

To obtain the representative executions, we applied the HRM [34] algorithm. Using ET as an anchor, we selected the 50th quantile (Q_{50}) of execution times across all runs as the representative ET for each TUA, then built a representative dataset containing all EMs for each isolated benchmark.

For every scenario, the HRM algorithm sorts each block of five EMs based on ET, allowing us to consider the quantile distribution of each EM. By reordering each block of EMs, we can select the same quantile Q_{50} of each EM metric and combine them to form a complete representative EM vector for each workload.

We then proceeded to build the final datasets from the contention experiments. For each scenario, we included the execution time in contention $TB_w(\tau)$ and tracked which benchmarks were active on each core (the tasks in the workload).

The contention dataset remained imbalanced because certain benchmarks experienced more pronounced overhead effects than others, which in turn led to more aggressive filtering process for those cases. To address this, we ensured that each TUA appeared an equal number of times and that each executed workload was represented five times. These repetitions capture the inherent variability in execution under identical contention conditions.

Lastly, we replaced each benchmark's name on each core with the corresponding representative EMs from the isolated experiments, thus forming our final training dataset.

We also considered the Task Order Invariance principle, which requires the model to produce the same prediction $\widehat{P\Delta}_w$, for EM_w regardless of how the cluster blocks are ordered [12]. It means that $\widehat{P\Delta}_w$ depends solely on which benchmarks are sharing resources in L2, not on the sequence in which they appear. To address this, we rearranged the order of the 3 clusters in contention according to their maximum ET in isolation. By fixing the order in the dataset, we ensure that the model's predictions do not depend on the specific permutation of the workload.

6.6. Training

The QRNN model was trained to predict the 90th quantile (Q_{90}) of factor contention, meaning that in the loss function (Eq. 5), the quantile parameter was set to $\gamma = 0.9$. The model leveraged EMs from isolated benchmarks as input features (Eq. 4), enabling it to capture the statistical distribution of execution times while accounting for potential contention effects.

Once trained, the QRNN can estimate the 90th quantile factor contention for a given set of execution conditions. Specifically, given a set of TUAs along with their contention benchmarks and resource allocation constraints, the model predicts a factor contention threshold. This threshold indicates that, for a given workload configuration, 90 % of execution times are expected to fall below the predicted value, ensuring a conservative yet reliable contention estimation.

7. Evaluation

This section presents the evaluation of the applicability of the QRNN-based techniques for the proposed analysis and optimization framework. Section 7.1 describes the experimental setups carried out. The experimental results are presented in Section 7.2. Finally, Section 7.3 discusses the obtained results focused on three main aspects: (1) Model applicability, (2) impact on the proposed framework, and (3) the effort comparison between measurement-based and estimation-based approaches.

7.1. Experimental setups

This section outlines the experiments conducted to evaluate the effectiveness of the model under multiple scenarios. Three main experiments were carried out under different setups, training specific models for each experiment.

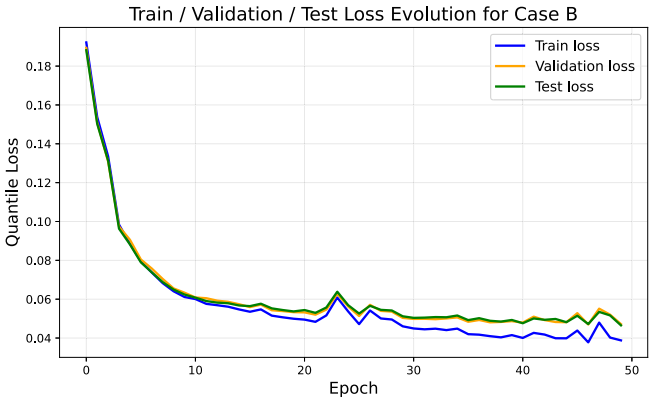


Fig. 6. Training/Validation/Test loss evolution for Case B.

The first experiment focuses on evaluating the QRNN model in a simple scenario, where contention is generated by a single core from each cluster outside the TUA's cluster (three contender setup). In this way, only one core is active per cluster. This scenario was chosen to provide a simplified approximation, allowing us to evaluate the model before moving to more complex configurations. For this purpose, a generic model was trained with workloads composed of all of the benchmarks described in Section 6.3, so inference was performed on tasks the model had already seen during training. In this initial evaluation, only *mem* variants of the benchmarks were used for the whole training and inferring process. We will refer to this experiment as **Test Case A**.

The objective of the second experiment is to evaluate how different memory access patterns affect the accuracy of the inference. In this case, a single model was trained with all benchmark variants, which includes data with varying lengths of memory (*dl1*, *l2*, and *mem* variants). Benchmark variants and different memory access patterns (i.e., sequential or random) were considered. As in the previous case, the model was trained with workloads composed of all of the benchmarks, and inference was performed on tasks the model had already seen during training. However, in this experiment, the model was trained for a full contender setup, where all the available cores complex may be generating contention, making the model sensitive to the complexity and heterogeneous design of the Carmel complex. This experiment will be denoted as **Test Case B**.

For both Test Cases A and B, the data were split into training (17 %), validation (2 %), and test (81 %) [40,41] sets using a stratified approach based on unique workloads, ensuring no overlap between subsets. Training was stable, with losses for both training and testing converging smoothly, as shown for Case B in Fig. 6, which corresponds to the more difficult task. The validation loss leveled off after around 30 epochs.

Finally, the last experiment aims to assess the behavior of the QRNN when tasks not seen during the training are inferred. To this end, the basic setup used in Test Case A has been reused: three contender setup with only *mem* variants of the benchmarks. Multiple models were trained, each one leaving out one specific benchmark from its training data. Then, each model is used to predict the contention values for the specific benchmark that has been left out during its training. This experiment will be referred to as **Test Case C**.

7.2. Experimental results

Test Case A - Simple approach

Test Case A aims to evaluate the model's behavior under a three contender setup where only *mem* variants are introduced. Results are presented using a scatter plot where each point represents a single prediction, in which the predicted contention factor is represented on the Y-axis, while the real contention factor is represented on the X-axis. As a reference, the $x = y$ line is drawn in red. Predictions lying on this line

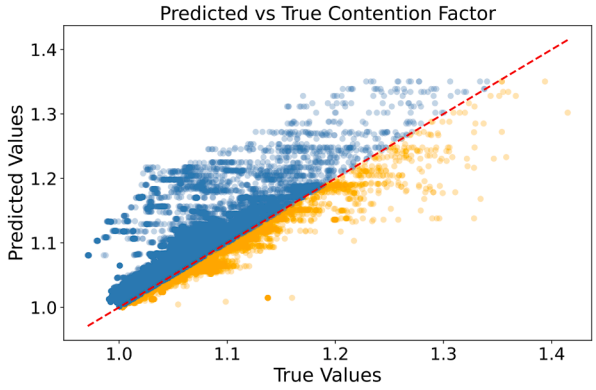


Fig. 7. Test Case A - Predicted vs true contention factor distribution for the three contender setup.

indicate that predictions matched the ground truth values. Points above the reference line represent overestimations (in blue), whereas points below the line represent underestimations (in orange).

Fig. 7 shows the evaluation of Test Case A. As shown, the model was able to accurately predict contention factor values while maintaining a low underestimation rate. From a total of 29,605 predictions, 88.19 % resulted in overestimations, whereas the remaining 11.81 % were underestimations. Notably, all estimations fell within an error threshold of 20 %.

Test Case B - Different memory access patterns

Test Case B assesses the accuracy of a model trained with different benchmark variants. For this evaluation, a total of 24,145 predictions were made.

First, we evaluated the model in a generic way, taking into account the predictions for all variants. The scatter plot of this evaluation is shown in Fig. 8a. From this plot it can be seen that the model tends to overestimate, accounting for 92.8 % of all predictions, while underestimations occurred only in 7.2 % of the predictions. To quantitatively evaluate the model's precision, we defined an error threshold of 20 % for both overestimations and underestimations. 73.47 % of predictions were overestimations within 20 % of the true value, while 6.44 % were underestimations within 20 % of the true value.

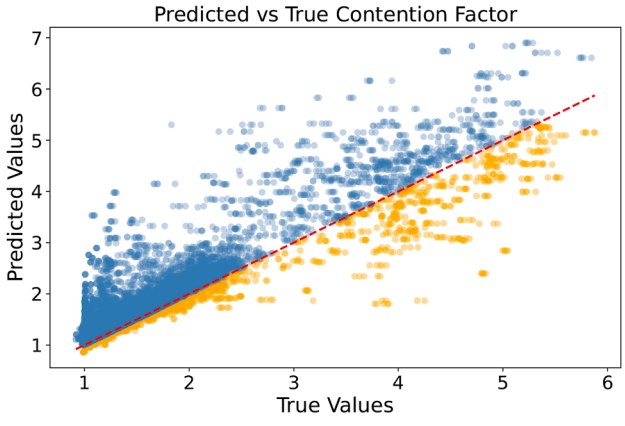
To assess the performance of the model under different memory access patterns, we have categorized the results into five distinct groups based on the benchmark characteristics: (1) Memory length variant in which tasks are forced to access the main memory (*mem*), (2) L2 cache sensitive variant (*l2*), (3) L1 data cache sensitive variant (*dl1*), (4) benchmarks with sequential memory access, and (5) benchmarks with random memory access (*rnd*). A breakdown of the results based on these categories is shown in Fig. 8b. The different categories are represented on the X-axis. The green and yellow bars represent the percentage of predictions that fell within the overestimations and underestimations error threshold of 20 %, respectively. The bars in blue and red represent the percentage of predictions outside the defined threshold.

Among the predictions for tasks with different memory lengths, the *mem* variant achieved the best results. This variant exhibited a low underestimation rate of 7.19 % and presented high accuracy for both over and underestimations. According to the *l2* variant, a degradation in accuracy can be observed. The underestimation rate slightly increased to 10.74 % and presented a higher tendency toward overestimation. The *dl1* variant, on the other hand, showed a reduction in underestimations, with no underestimation remaining within the 20 % error threshold. This is because, in most cases, no contention occurred as the CPU does not compete for the L1 data cache with other cores. Thus, since the contention factor cannot be less than 1, the model was implicitly able to avoid underestimation under this scenario. However, even in the absence of real contention, the model maintains its tendency to overestimate. Consequently, the model presented a significant drop in accuracy, with only 42.24 % of predictions falling within the defined error threshold for overestimations.

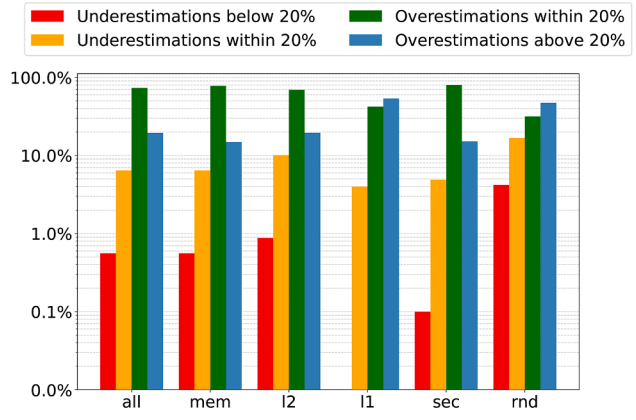
The next step is to analyze the results based on benchmarks with sequential or random memory access. For benchmarks with sequential memory access (denoted in Fig. 8b as *sec*), the model achieves very high accuracy, with a low underestimation rate of 5.4 %. In this scenario, 79.61 % of predictions fell within the overestimation error threshold, and only a 0.08 % of underestimations remained outside this limit, indicating strong reliability. This percentage of underestimations corresponds to the *l2* variants where the underestimation rate was higher. In contrast, the model's performance deteriorates significantly for benchmarks with random memory access (denoted as *rnd*). The underestimation rate increased to 21.4 %, and accuracy dropped, with only 31.6 % of predictions falling within the overestimation's 20 % error threshold and with the highest rate of underestimations, 4.4 % of predictions, below the defined error bounds.

Test Case C - Unseen tasks

Test Case C introduces multiple models, each with a particular benchmark not seen during its training process. In this evaluation, a notable performance degradation was observed that greatly depends on the type of benchmark under analysis. Fig. 9 illustrates three representative examples. In the scenario presented in Fig. 9a, the model continued

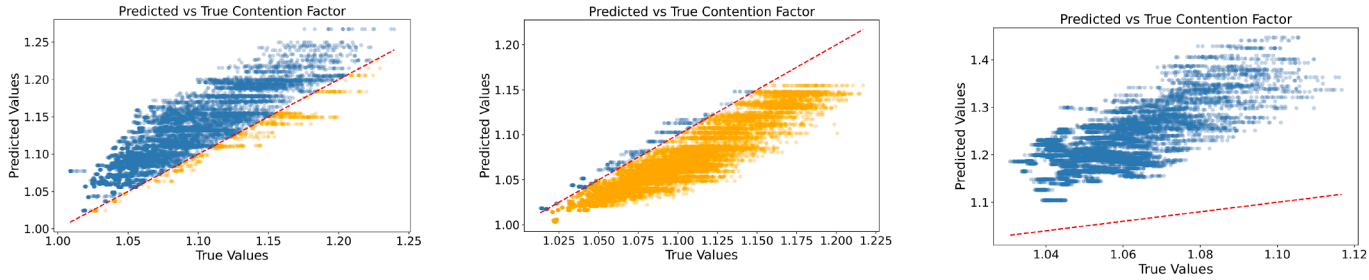


(a) Predicted vs true contention factor distribution for full contender setup.



(b) Breakdown of the results categorized by the memory access pattern.

Fig. 8. Test Case B - Results for benchmarks with different memory access patterns under full contender setup.



(a) Results for inferring excluded benchmark ldr_downsampling_f4.

(b) Results for inferring excluded benchmark ldr_filtering.

(c) Results for inferring excluded benchmark rnd_acc.

Fig. 9. Test Case C - Three representative cases of predicted vs. true contention factor for the three contender setup, inferring over a benchmark excluded from the training set.

to predict accurately and with a low underestimation rate. However, in scenarios Fig. 9b and Fig. 9c, the models systematically underestimated and overestimated the contention factors, respectively. These results reveal a potential limitation: while the model is accurate within its training domain, it struggles to make correct predictions for tasks that are not present during training.

7.3. Discussion

7.3.1. Model applicability

Concerning the applicability of the QRNN-based techniques within the proposed analysis and optimization framework, two main considerations can be derived from the obtained results.

In the first experiment, **Test Case A**, a simple scenario was assessed with just one interfering core per cluster, while the inference is performed on a task seen during training. Under this scenario, the model is able to produce accurate results.

Subsequently, in **Test Case B**, we evaluated a model trained with different benchmark variants under a full contender setup (i.e., all cores). The results reveal that the performance of the model could vary significantly depending on the type of workload. This indicates that the model struggles to generalize across tasks with different memory access patterns. This may lead to two possible approaches in order to integrate the prediction model into the proposed optimization framework. The first approach would consist of building task-specific models, each trained for task groups with a specific memory access pattern. This could lead to higher accuracy, but at the cost of increased complexity to prepare and train the models. In contrast, using a single generic model may reduce the development complexity at the potential expense of reduced accuracy. Each approach presents a trade-off between precision and the effort required for training and developing the necessary models.

In the last experiment, **Test Case C**, we inferred the contention of different tasks that were not seen during the training phase. We found that in this case, the models produced less accurate predictions. This suggests that, for the integration of the model into the proposed optimization framework, it would be necessary to train a system-specific model tailored not only to the platform but also to the whole task set that composes the system under analysis. It is important to note that the proposed optimization framework is intended to be used during the design phase of the system, in which the whole task set must already be known and characterized to perform other basic activities, such as the schedulability analysis.

The addition of new tasks could be handled by first evaluating the prediction performance of the existing model on that new task. If the results are satisfactory, the task can be safely integrated into the analysis and optimization process. Only in cases where a considerable performance degradation is observed would it be necessary to retrain a system-specific model. For example, in the case represented in Fig. 9b, where the model showed a tendency to underestimate, retraining the model including this specific task would be necessary. The key advantage presented by this approach is that the effort to collect the additional necessary data and train the model would only need to be done once. After retraining, the model would be able to handle any possible system reconfiguration.

In conclusion, these results indicate the feasibility of integrating QRNN-based prediction techniques in an analysis and optimization process. For a given real-time system, an initial model should be trained and evaluated to determine if it would be necessary to use a task-specific or a system-specific approach. With this, if necessary, new models could be trained and reevaluated before a final implementation is established.

Although the presented framework is directly valid for mixed-critical systems, its application in high-critical systems may be compromised due to the possibility of some occasional underestimations. In this scenario, the validity of the solution will require direct verification of the execution times under the final configuration proposed by the analysis and optimization framework and, if necessary, performing a sensitivity analysis on this final solution to assess its robustness.

7.3.2. Impact on the proposed framework

Although most of the underestimated predictions fell within an error threshold of 20 %, some cases deviate from this margin, making the model unable to bound underestimations. This behavior can have significant implications in a real-time system, as an underestimated contention factor could lead to deadline misses. Nevertheless, adopting a conservative strategy could mitigate this effect in the proposed framework. For each task, as the worst-case scenario contention is considered (i.e., the one given by the workload with a higher contention factor), the model can still be effective even if predictions are not accurate in absolute terms.

In this context, it becomes crucial that the models preserve a correlation with the true values. If the model consistently assigns estimated contention factors such that it estimates more contention when actual contention is higher, and less when it is lower, then the worst-case contention scenario can still be correctly identified.

Fig. 10 represents the behavior described above. In the blue line, we can see all the true values of the full contender setup, ordered from the highest to the lowest contention factor. The predicted values are represented in orange. Finally, the red line represents the prediction's tendency by considering median values. These median predicted values were calculated using fixed-size bins. We can observe notable noise in prediction values because, as we have seen previously, the model tends to overestimate differently depending on the benchmark. However, predictions tend to maintain a consistent relation with true values.

To characterize this correlation, we compute the Pearson correlation coefficient for each different benchmark group described in Section 7. This analysis shows the correlation between predicted and true contention factors. Table 2 shows the correlation obtained for each benchmark variant and for the whole set. In general, a strong correlation is observed above 80 % across all cases except the *dl1* variant. In this case,

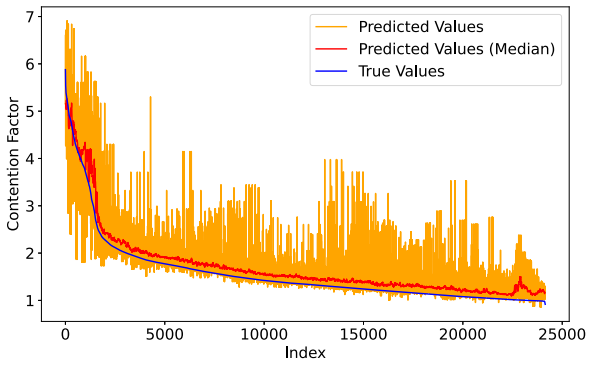


Fig. 10. Correlation between predicted and true values.

Table 2
Pearson correlations by benchmark group.

TUA	Correlation (%)
mem	92.49
l2	80.46
l1	38.01
rnd	87.12
sec	90.76
all	92.31

as shown in the experimental results, the model was not able to correctly learn the L1 memory access behavior.

7.3.3. Effort comparison - Measurement-based vs estimation-based approaches

For the proposed analysis and optimization framework, two alternatives were considered to obtain new WCET estimates under different system configurations: (1) performing empirical measurements on the real platform for each configuration or, (2) using prediction techniques like the one proposed in this work. In this section, we estimate the potential time and effort costs by adopting the estimation-based approach compared to the measurement-based one.

We define the total time required to complete the optimization process deploying tests on the real platform as:

$$T_{Deploy} = T_M \times N \times I \tag{6}$$

where T_M represents the average time required to measure the WCET estimates for a single workload, N denotes the number of potential workloads that are possible under a certain configuration proposed by the optimization tool (for this evaluation, we assume a fixed representative value), and I is the number of iterations needed by the framework to get an optimal configuration.

In contrast, the time required by the estimation-based approach, considering not only the effort needed to obtain new WCET estimates but also the cost of preparing the prediction model, can be represented as:

$$T_{Estim} = T_{DS} + T_{Train} + T_{Infer} \times N \times I \tag{7}$$

where T_{DS} represents the time needed to obtain the Data Set, T_{Train} is the training time for the model, and T_{Infer} represents the average time needed to obtain a single prediction. This assumes that a single model is trained. If task-specific models are used instead, it would be necessary to add to the needed time to train the different models.

To provide an illustrative view of this trade-off, we present an estimated analysis in which the number of optimization iterations increases progressively. Table 3 summarizes the approximated time values used for the model presented in Test Case B. As a representative N, we consider a system composed of 24 tasks (corresponding to the total number of benchmarks in Test Case B) uniformly distributed across the eight cores, resulting in 3 tasks statically assigned per core. Assuming that

Table 3
Required time estimations for Test Case B.

Value	Required time
T_M	30 seconds
T_{DS}	200 hours
T_{Train}	45 minutes
T_{Infer}	0.3 milliseconds

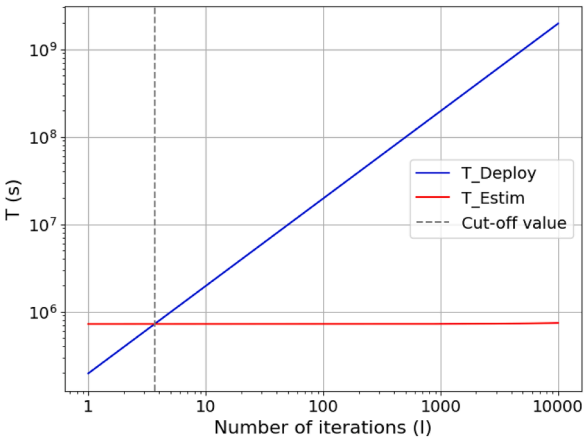


Fig. 11. Effort trade-off between measured-based and estimation-based approaches.

cores are fully available for the execution of the assigned tasks, the maximum number of possible workload combinations in this scenario is 3^8. This value is used as the representative N in our illustrative analysis.

Fig. 11 shows the trade-off between the measurement-based and the estimation-based approaches, with times in seconds (note that logarithmic scale is used). While for the measurement-based approach, the required time increases significantly with the number of iterations, the estimation-based approach experiences a slight increase. This is because the main effort is made at the beginning when preparing the dataset and training the model, and then, the model remains valid for inferring every possible combination within the system with a low effort. In the presented scenario, only four optimization iterations throughout the system's life cycle are needed to benefit from the estimation-based approach. The results shown are valid only for a system with the features presented in this evaluation, which contains few tasks. However, for a real-world large-scale system, which might contain hundreds of tasks, where the combinations can grow exponentially, the measurement-based approach would be even less feasible. This effect can be further amplified in modern MPSoCs by adding multiple types of processing resources, with different memory sizes operating at different bandwidths, and varying cache hierarchies. In contrast, the estimation-based approach may still efficiently scale under this scenario since the model needs to be trained only once and be reused for every possible combination.

8. Conclusions and future work

In this work, we proposed an analysis and optimization framework that enables dynamic WCET re-evaluation based on the system configuration to reflect the impact of memory contention on heterogeneous platforms. Unlike traditional methodologies, where the WCETs are assumed to be known and fixed during the entire process, the proposed methodology is based on generating new WCET estimations, taking into account the memory contention for each specific configuration. Although not considered in this work, there are several mechanisms, such as time partitioning, that not only allow us to mitigate the memory

contention, but also to control the task subsets that could be executed concurrently in different processors.

To derive the new WCETs, we proposed the application of QRNN-based techniques, which efficiently estimate the memory contention that a task may experience under certain contention scenarios based on the measured Event Monitors (EMs). This work evaluated the applicability of these techniques for a framework such as the one proposed in this work.

The presented evaluation reveals key insights into the applicability of QRNN-based techniques during the design phase of real-time systems. Results showed that the model's accuracy can vary significantly depending on the memory access patterns of the tasks, such as tasks with random memory access or those with different memory footprints. This may lead to two potential strategies to integrate QRNNs into the proposed optimization framework: (1) opting for a generic model that simplifies the development but may have a reduced prediction accuracy, or (2) developing task-specific models for higher accuracy. Each strategy offers a distinct trade-off between accuracy and development complexity.

Moreover, experiments demonstrated that models trained on a specific task set struggled to make accurate predictions when applied to unseen tasks. This highlights the potential need for system-specific models trained with the whole task-set. When new tasks must be added into the system, the performance of the model on that task can be evaluated to determine whether retraining a system-specific model is needed. The process involves a one-time data collection and training for every new task or task set. Once retrained, it offers the advantage of being able to handle any system configuration that may be tested during the optimization phase.

Although the application of QRNN can lead to underestimated values, it can still be effective when using a conservative strategy, considering each task's worst-case contention scenario. The model showed the ability to preserve a strong correlation with true contention values. This behavior makes it possible to consistently identify the worst-case contention scenarios for each task. Experimental results support this finding, showing an overall correlation of 92.31 %.

In addition, an analysis has been conducted to evaluate the effort needed for the estimation process itself. A comparative evaluation between measurement-based and estimation-based approaches showed the scalability advantages of the estimation-based approach. While empirical measurements become increasingly time-consuming as the number of workload combinations and optimization iterations grows, the estimation-based approach incurs most of its cost upfront during dataset and model preparation and training. This means that for large systems, an estimation-based approach is preferable, as you can cover the entire life cycle of a system after an initial effort, whereas the measurement-based approach becomes highly costly or even unfeasible.

Finally, this work demonstrates that imprecise techniques such as the QRNNs evaluated in this paper can be safely used to estimate WCETs in the optimization process of real-time systems, assuming that, if necessary, validation of the final solution(s) can be performed. Therefore, this work opens an opportunity for these techniques to be used on modern heterogeneous platforms, even in critical systems where their usefulness may be questionable, since their results may not fully guarantee the worst-case conditions.

As future work, we propose to validate the strategy in a real smart mobility application, which is currently being implemented and modeled. Moreover, in this work, only the multicore contention has been considered, even though the GPUs and other accelerators represent an additional contention point. The evaluation of the effects of these additional resources has also been planned. Finally, future work includes addressing the development of an implementation of the proposed framework, integrating the WCET estimation based on QRNNs into other analysis and optimization tools such as MAST [37], with which there is already previous experience in adaptation to other models or integration of tools following an MDE strategy [42].

CRediT authorship contribution statement

Iosu Gomez: Writing – review & editing, Writing – original draft, Software, Investigation, Conceptualization; **David Fonts:** Writing – review & editing, Writing – original draft, Software, Investigation, Conceptualization; **Sergi Vilardell:** Writing – review & editing, Writing – original draft, Software, Investigation, Conceptualization; **Unai Díaz De Cerio:** Writing – review & editing, Supervision, Conceptualization; **Juan M. Rivas:** Writing – review & editing, Supervision, Conceptualization; **Enrico Mezzetti:** Writing – review & editing, Supervision, Conceptualization; **J. Javier Gutiérrez:** Writing – review & editing, Supervision, Conceptualization; **Francisco J. Cazorla:** Writing – review & editing, Supervision, Conceptualization.

Data availability

The authors do not have permission to share data.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was partially funded by MICIU/ AEI /10.13039/501100011033 and FEDER, UE under grants PID2021-124502OB-C42 and PID2021-124502OB-C44 (PRESECREL), and also PID2024-155230OB-C41 and PID2024-155230OB-C44 (Re-InITS).

Appendix A. Core and uncore events in Jetson AGX Xavier

This appendix presents the core and uncore events available for the Jetson AGX Xavier, as shown in Table A.4 and Table A.5 respectively. The information has been obtained from the NVIDIA Technical Reference Manual for the Xavier series SoC [43].

Table A.4
Core events in Jetson AGX Xavier.

Event	Description	Event	Description
SW_INCR	Instruction architecturally executed, software increment	INST_RETIRED	Instruction architecturally executed
EXC_TAKEN	Exception taken	EXC_RETURN	Instruction architecturally executed, exception return
CID_WRITE_RETIRED	Instruction architecturally executed, Write to CONTEXTIDR	TTBR_WRITE_RETIRED	Instruction architecturally executed, Write to TTBR
L1D_CACHE_LD	Level-1 Data Cache access, Read	L1D_CACHE_ST	Level-1 Data Cache access, Write
CPU_CYCLES_DUAL_EXEC	Cycles in dual execution mode	CPU_CYCLES_DUAL_EXEC_ELIGI	Cycles dual execution mode is eligible to execute
L1I_CACHE_REFILL	Level-1 Instruction Cache refill	L1I_TLB_REFILL	Level-1 instruction TLB refill
L1D_CACHE_REFILL	Level-1 Data Cache refil	L1D_TLB_REFILL	Level-1 data TLB refill
BR_MIS_PRED	Mispredicted or not predicted branch speculatively executed	CPU_CYCLES	CPU cycles
BR_PRED	Predictable branch speculatively executed	MEM_ACCESS	Data memory access
L1I_CACHE	Level-1 Instruction Cache access	L1D_CACHE_WB	Level-1 Data Cache Write-back
MEMORY_ERROR	Local memory error	STALL_FRONTEND	No operation issued due to the frontend
STALL_BACKEND	No operation issued due to the backend	L1D_CACHE_REFILL_LD	Level-1 Data Cache refill, Read
L1D_CACHE_REFILL_ST	Level-1 Data Cache refill, Write	L1D_CACHE_WB_VICTIM	Level-1 Data Cache Write-back, victim
L1D_CACHE	Level-1 Data Cache access		

Table A.5
Uncore events in Jetson AGX Xavier.

Event	Description	Event	Description
L2D_CACHE	Level-2 Data Cache access	L2D_CACHE_REFILL	Level-2 Data Cache refill
L2D_CACHE_WB	Attributable Level-2 Data Cache Write-back	BUS_ACCESS	Bus access
BUS_CYCLES	Bus cycle	L3D_CACHE_ALOCATE	Level-3 Data Cache allocation without refill
L3D_CACHE_REFILL	Level-3 Data Cache refill	L3D_CACHE	Level-3 Data Cache access
L3D_CACHE_WB	Level-3 Data Cache Write-Back		

Table A.6
List of used custom benchmarks.

Benchmark name	Description	Benchmark name	Description
rnd_acc	Performs 1M random memory accesses.	rnd_acc_2	Performs 2M random memory accesses. Has a *dl1* and a *l2* variant.
matrix_sum	Computes the sum of two matrices.	matrix_trans	Transposes a given matrix, which involves switching its rows and columns.
matrix_trans_mltpl	First transposes a matrix and then multiplies it with another matrix.	matrix_mmb	Computes a matrix multiplication.
v_add_int	Computes the sum of two vectors. Has a *dl1* variant.	v_mltpl	Computes a vector multiplication. Has a *l2* variant.
v_mltpl_add	Computes a vector multiplication followed by addition.	v_rect	Performs the rectifier (ReLU) function to input data.
v_scale	Scales the values of a vector by a constant factor.	genetic_algorithm	Railway application based on a genetic algorithm, used as a use case in [35].
ldr_downsampling_f2	Reduces the resolution of LIDAR point-cloud data by a factor of 2. Has a *l1* variant.	ldr_downsampling_f4	Reduces the resolution of LIDAR point-cloud data by a factor of 4. Has a *l2* variant.
ldr_downsampling_f4	Reduces the resolution of LIDAR point-cloud data by a factor of 10.	ldr_filtering	Applies filtering to LIDAR point-cloud to eliminate ground and distant objects. Has a *dl1* and a *l2* variant.

Appendix B. Custom benchmarks

This appendix presents the set of custom benchmarks used in this study. Table A.6 provides a description of each benchmark and indicates which ones have *dl1* or *l2* variants on their data footprint. The *mem* variant is assumed by default. The input data of these benchmarks is composed of random numbers. Regarding the memory access pattern, all the benchmarks access the memory sequentially except the rnd_acc and rnd_acc_2 benchmarks, which perform random memory accesses.

The code for these benchmarks has been made publicly available, including the measured EMs, memory contention measurements, and the kernel module to access core and uncore PMUs: https://github.com/xavier-pmu-tools/fgcs_carmel_bench_and_data.

References

[1] Building the European Cloud, Edge & IoT Continuum for business and research,. https://eucloudedgeiot.eu/.

[2] M. Andreozzi, G. Gabrielli, B. Venu, G. Travaglini, Industrial challenge 2022: a high-Performance real-Time case study on arm, Leibniz Int. Proc. Inform., LIPIcs 231 (2022). https://doi.org/10.4230/LIPIcs.ECRTS.2022.1

[3] S. Moulik, R. Devaraj, A. Sarkar, HETERO-SCHED: A low-Overhead heterogeneous multi-core scheduler for real-Time periodic tasks, in: 2018 IEEE 20Th International Conference on High Performance Computing and Communications (HPCC), IEEE, 2018, pp. 659–666. https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00117

[4] S. Moulik, Z. Das, R. Devaraj, S. Chakraborty, SEAMERS: A semi-partitioned energy-Aware scheduler for heterogeneous multicore real-time systems, J. Syst. Archit. 114 (2021) 101953. https://doi.org/10.1016/j.sysarc.2020.101953

[5] A. Fernández de Lecea, M. Hassan, E. Mezzetti, J. Abella, F.J. Cazorla, Improving timing-Related guarantees for main memory in multicore critical embedded systems, in: 2023 IEEE Real-Time Systems Symposium (RTSS), 2023, pp. 265–278. https://doi.org/10.1109/RTSS59052.2023.00031

[6] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, R. Rajkumar, Bounding memory interference delay in COTS-based multi-core systems, in: 2014 IEEE 19Th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014, pp. 145–154. https://doi.org/10.1109/RTAS.2014.6925998

[7] C. Maiza, H. Rihani, J.M. Rivas, J. Goossens, S. Altmeyer, R.I. Davis, A survey of timing verification techniques for multi-Core real-Time systems, ACM Comput. Surv. 52 (3) (2019). https://doi.org/10.1145/3323212

[8] D. Hardy, I. Puaut, Y. Sazeides, Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults, in: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 91–96.

[9] B. Lesage, D. Griffin, S. Altmeyer, L. Cucu-Grosjean, R.I. Davis, On the analysis of random replacement caches using static probabilistic timing methods for multi-path programs, Real-Time Syst. 54 (2) (2018) 307–388. https://doi.org/10.1007/s11241-017-9295-2

[10] A. Burns, R.I. Davis, A survey of research into mixed criticality systems, ACM Comput. Surv. 50 (6) (2017). https://doi.org/10.1145/3131347

[11] A.J. Cannon, Quantile regression neural networks: implementation in r and application to precipitation downscaling, Comput. Geosci. 37 (9) (2011) 1277–1284. https://doi.org/10.1016/j.cageo.2010.07.005

[12] A. Brando, I. Serra, E. Mezzetti, J. Abella, F.J. Cazorla, Using quantile regression in neural networks for contention prediction in multicore processors, in: 34Th Euromicro Conference on Real-Time Systems (ECRTS 2022), 231 of *Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany*, 2022, pp. 4:1–4:25. https://doi.org/10.4230/LIPIcs.ECRTS.2022.4

[13] S. Altmeyer, E. André, S. Dal Zilio, L. Fejoz, M.G. Harbour, S. Graf, J.J. Gutiérrez, R. Henia, D. Le Botlan, G. Lipari, J. Medina, N. Navet, S. Quinton, J.M. Rivas, Y. Sun, From FMTV to WATERS: lessons learned from the first verification challenge at ECRTS, in: A.V. Papadopoulos (Ed.), 35Th Euromicro Conference on Real-Time Systems (ECRTS 2023), 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2023, pp. 19:1–19:18. https://doi.org/10.4230/LIPIcs.ECRTS.2023.19

[14] D. Dasari, V. Nelis, B. Andersson, WCET Analysis considering contention on memory bus in COTS-based multicores, in: Etfa2011, 2011, pp. 1–4. https://doi.org/10.1109/ETFA.2011.6059176

[15] H. Yun, R. Pellizzon, P.K. Valsan, Parallelism-Aware memory interference delay analysis for COTS multicore systems, in: 2015 27Th Euromicro Conference on Real-Time Systems, 2015, pp. 184–195. https://doi.org/10.1109/ECRTS.2015.24

[16] M. Hassan, R. Pellizzoni, Analysis of memory-Contention in heterogeneous COTS MPSocs, in: M. Völp (Ed.), 32Nd Euromicro Conference on Real-Time Systems (ECRTS 2020), 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 23:1–23:24. https://doi.org/10.4230/LIPIcs.ECRTS.2020.23

[17] D. Dasari, B. Andersson, V. Nelis, S.M. Petters, A. Easwaran, J. Lee, Response time analysis of COTS-Based multicores considering the contention on the shared memory bus, in: 2011IEEE 10Th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 1068–1075. https://doi.org/10.1109/TrustCom.2011.146

[18] J. Xiao, S. Altmeyer, A. Pimentel, Schedulability analysis of non-preemptive real-Time scheduling for multicore processors with shared caches, in: 2017 IEEE Real-Time Systems Symposium (RTSS), 2017, pp. 199–208. https://doi.org/10.1109/RTSS.2017.00026

[19] S. Altmeyer, R.I. Davis, L. Indrusiak, C. Maiza, V. Nelis, J. Reineke, A generic and compositional framework for multicore response time analysis, in: Proceedings of the 23Rd International Conference on Real Time and Networks Systems, RTNS '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 129-138. https://doi.org/10.1145/2834848.2834862

[20] R. Mancuso, R. Pellizzoni, M. Caccamo, L.R. Sha, H. Yun, WCET(M) estimation in multi-core systems using single core equivalence, 2015 27th Euromicro Conference on Real-Time Systems (2015) 174–183.

[21] J. Arora, C. Maia, S.A. Rashid, G. Nelissen, E. Tovar, Schedulability analysis for 3-phase tasks with partitioned fixed-priority scheduling, J. Syst. Archit. 131 (2022) 102706. https://doi.org/10.1016/j.sysarc.2022.102706

[22] A. Bonenfant, D. Claraz, M. de Michiel, P. Sotin, Early WCET prediction using machine learning, in: J. Reineke (Ed.), 17Th International Workshop on Worst-Case Execution Time Analysis (WCET 2017), 57 of *Open Access Series in Informatics (OASIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017, pp. 5:1–5:9. https://doi.org/10.4230/OASIcs.WCET.2017.5

[23] T. Huybrechts, S. Mercelis, P. Hellinckx, A new hybrid approach on WCET analysis for real-Time systems using machine learning, in: F. Brandner (Ed.), 18Th International Workshop on Worst-Case Execution Time Analysis (WCET 2018), 63 of *Open Access Series in Informatics (OASIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018, pp. 5:1–5:12. https://doi.org/10.4230/OASIcs.WCET.2018.5

[24] J. Zhao, H. Cui, J. Xue, X. Feng, Predicting cross-Core performance interference on multicore processors with regression analysis, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1443–1456. https://doi.org/10.1109/TPDS.2015.2442983

[25] S. Ren, L. He, J. Li, Z. Chen, P. Jiang, C.-T. Li, Contention-aware prediction for performance impact of task co-running in multicore computers, Wireless Networks 28 (3) (2022) 1293–1300. https://doi.org/10.1007/s11276-018-01902-7

[26] A.N. Amalou, E. Fromont, I. Puaut, CAWET: Context-Aware worst-Case execution time estimation using transformers, in: 35Th Euromicro Conference on Real-Time Systems (ECRTS 2023), 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2023, pp. 7:1–7:20. https://doi.org/10.4230/LIPIcs.ECRTS.2023.7

[27] V. Kumar, B. Ranjbar, A. Kumar, Utilizing machine learning techniques for worst-Case execution time estimation on GPU architectures, IEEE Access 12 (2024) 41464–41478. https://doi.org/10.1109/ACCESS.2024.3379018

[28] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, R. Rajkumar, Bounding and reducing memory interference in COTS-based multi-core systems, Real-Time Syst. 52 (3) (2016) 356–395. https://doi.org/10.1007/s11241-016-9248-1

[29] S. Skalistis, A. Simalatsar, Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, 2017, pp. 752–757. https://doi.org/10.23919/DATE.2017.7927090

[30] J.M. Aceituno, A. Guasque, P. Balbastre, J. Simó, A. Crespo, Interference-Aware schedulability analysis and task allocation for multicore hard real-Time systems, Electronics 11 (9) (2022). https://doi.org/10.3390/electronics11091313

[31] J.M. Rivas, J.J. Gutierrez, J.C. Palencia, M.G. Harbour, Schedulability analysis and optimization of heterogeneous EDF and FP distributed real-Time systems, in: 2011 23Rd Euromicro Conference on Real-Time Systems, 2011, pp. 195–204. https://doi.org/10.1109/ECRTS.2011.26

[32] Q. Zhu, H. Zeng, W. Zheng, M.D.I. Natale, A. Sangiovanni-Vincentelli, Optimization of task allocation and priority assignment in hard real-time distributed systems, ACM Trans. Embed. Comput. Syst. 11 (4) (2013). https://doi.org/10.1145/2362336.2362352

[33] J.M. Rivas, J.J. Gutiérrez, M. González Harbour, A supercomputing framework for the evaluation of real-time analysis and optimization techniques, J. Syst. Software 124 (2017) 120–136. https://doi.org/10.1016/j.jss.2016.11.010

[34] S. Vilardell, I. Serra, R. Santalla, E. Mezzetti, J. Abella, F. Cazorla, HRM: Merging hardware event monitors for improved timing analysis of complex MPSocs, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 39 (2020) 1–1. https://doi.org/10.1109/TCAD.2020.3013051

[35] I. Gomez, U. Díaz de Cerio, J. Parra, J.M. Rivas, J.J. Gutiérrez, M.G. Harbour, Using MAST for modeling and response-time analysis of real-time applications with GPUs, J. Syst. Archit. 157 (2024) 103300. https://doi.org/10.1016/j.sysarc.2024.103300

[36] N. Capodieci, R. Cavicchioli, I.S. Olmedo, M. Solieri, M. Bertogna, Contending memory in heterogeneous socs: evolution in NVIDIA tegra embedded platforms, in: 2020 IEEE 26Th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2020, pp. 1–10. https://doi.org/10.1109/RTCSA50079.2020.9203722

[37] S. engineering, r.-t. g. .U. de Cantabria, MAST - Modeling and Analysis Suite for real-Time applications,. https://mast.unican.es/.

[38] N. Corporation, Jetson AGX Xavier Series. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[39] M. Kerris, Linux manual page - perf_event_open(2). https://man7.org/linux/man-pages/man2/perf_event_open.2.HTML.

[40] H. Drucker, C.J. Burges, L. Kaufman, A. Smola, V. Vapnik, Support vector regression machines, in: Advances in Neural Information Processing Systems, 9, 1996. https://doi.org/10.5555/2998981.2999003

[41] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, J. Snoek, Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift, in: Advances in Neural Information Processing Systems, 32, 2019.

[42] J.M. Rivas, J.J. Gutiérrez, M. Aldea, C. Cuevas, M. González Harbour, J.M. Drake, J.L. Medina, L. Rioux, R. Henia, N. Sordon, An experience integrating response-Time analysis and optimization with an MDE strategy, in: P. Milazzo, D. Varró, M. Wimmer (Eds.), Software Technologies: Applications and Foundations, Springer International Publishing, Cham, 2016, pp. 303–316. https://doi.org/10.1007/978-3-319-50230-4_23

[43] N. Corporation, Xavier Series SoC Technical Reference Manual, 2018. ID: DP-09253-002 Version: 1.1.