Uncertainty-Aware Resource Allocation for Multi-Path Programs with In-Kernel Predictions

Abigail Eisenklam[1]
University of Pennsylvania,
Philadelphia, PA, USA

Carlos A. Montenegro G.[1]
University of California,
Santa Cruz, CA, USA

Xian Wang
University of Pennsylvania,
Philadelphia, PA, USA

Yifan Cai
University of Pennsylvania,
Philadelphia, PA, USA

Robert Gifford
University of Pennsylvania,
Philadelphia, PA, USA

Linh Thi Xuan Phan
University of Pennsylvania,
Philadelphia, PA, USA

Ricardo G. Sanfelice
University of California,
Santa Cruz, CA, USA

Abstract

Predictable timing on multicore systems requires careful management of shared resources such as the last-level cache and memory bandwidth. This paper presents MPORA, an uncertainty-aware dynamic resource allocation framework for multi-path, input-dependent real-time tasks on multicore platforms. MPORA models each job as a discrete-time dynamical system that captures execution dynamics and resource-dependent performance indicators. At runtime, MPORA monitors job execution states and predicts short-term instruction rates and remaining execution times under candidate allocations using predictive models trained offline. It then solves a receding-horizon optimization problem to compute resource allocations that maximize system-wide progress while meeting job deadlines. To address prediction uncertainty, MPORA integrates weighted conformal prediction into the optimization formulation, enabling uncertainty-aware deadline constraints. We implement MPORA as a Linux kernel module with microsecond-scale inference overhead. Experimental results on SPEC CPU benchmarks show that MPORA delivers accurate predictions under unseen inputs and distribution shifts with low overhead, while improving schedulability and response times over existing methods.

2012 ACM Subject Classification Computer systems organization → Real-time operating systems

Keywords and phrases multicore, resource allocation, optimal control, learning, multi-path programs

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2026.17

Supplementary Material *Software (Source Code)*: `https://github.com/phan-lab/MPORA` [19]
archived at `swh:1:dir:effbb0b80f77dea1364355a09ce8c305aebc15b2`
Software (ECRTS 2026 Artifact Evaluation approved artifact):
`https://doi.org/10.4230/DARTS.12.2.9`

Funding Partially supported by NSF Grants no. CNS-1955670, CNS-2039054, CNS-2111688, and CCF-2326606, by AFOSR Grants nos. FA9550-23-1-0145, FA9550-23-1-0313, and FA9550-23-1-0678, by AFRL Grant nos. FA8651-22-1-0017 and FA8651-23-1-0004, by ARO Grant no. W911NF-20-1-0253, by DoD Grant no. W911NF-23-1-0158, and by UC Alianza MX Grant no. SRPUCMX24-01.

[1] Equal contribution.

© Abigail Eisenklam, Carlos A. Montenegro G., Xian Wang, Yifan Cai, Robert Gifford, Linh Thi Xuan Phan, and Ricardo G. Sanfelice; licensed under Creative Commons License CC-BY 4.0

38th European Conference on Real-Time Systems (ECRTS 2026).
Editor: Angeliki Kritikakou; Article No. 17; pp. 17:1–17:26
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

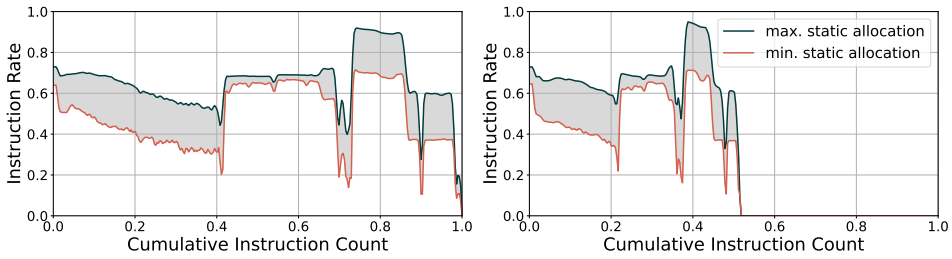


Figure 1 Profiles for SPEC benchmark *omnetpp* with a large (left) vs. medium (right) input. The shaded region represents possible rates under dynamic allocation. Axes are normalized to [0, 1].

1 Introduction

Modern real-time systems increasingly run on multicore platforms to meet growing computational demands. However, sharing resources – such as last-level caches and memory bandwidth – among cores can cause timing interference: concurrently executing tasks may contend for the same resources, leading to unpredictable execution times. Careful allocation of these resources is therefore critical to ensure real-time performance.

State-of-the-art multicore resource allocation strategies, however, are constrained by static assumptions or design choices. Some methods fix resource assignments at design time [61, 60, 44, 45, 55] and assume static resource demands, often leaving resources underutilized. Others adjust allocations dynamically but still rely on predetermined execution paths or fixed inputs, overlooking the inherent variability of real-world workloads [48, 50, 26, 27, 20, 25].

In reality, the timing behaviors and resource demands of real-time tasks fluctuate across program phases and depend on inputs, code paths, and scheduling decisions [16] (c.f. Fig. 1). To adapt to these changing conditions, we propose a more *responsive* approach: instead of provisioning for worst-case behavior or assuming fixed paths, the system continuously monitors execution state, predicts execution progress under candidate allocations, and adjusts resources to optimize utilization and response time under deadline constraints.

Contributions. This paper presents MPORA, an uncertainty-aware dynamic resource allocation framework for multicore real-time systems targeting input-dependent multi-path programs. MPORA models each job as a discrete-time dynamical system whose state captures execution features such as current progress and resource-usage indicators. Using profiling data collected across diverse inputs and allocations, MPORA learns regression models that predict short-term execution rates and remaining execution times under candidate allocations.

At runtime, MPORA employs a receding-horizon, model predictive control (MPC)-like approach: at each allocation step, it solves a finite-horizon optimization problem that selects resource assignments to optimize predicted instructions retired and job response times. It recomputes allocations often, enabling adaptation to system state changes. To account for prediction errors, MPORA provides a novel integration with weighted conformal prediction (CP) that provides uncertainty-aware deadline constraints in our dynamic resource allocation.

Realizing MPORA in practical real-time systems requires addressing three systems challenges: low-overhead execution-state characterization, microsecond-scale model inference and uncertainty estimation, and efficient solution of the finite-horizon allocation problem within the kernel. Thus, MPORA is implemented as a Linux kernel module using fixed-point gradient boosting regression trees (GBRT) for prediction.

We evaluate MPORA on benchmarks from the SPEC 2017 Intspeed suite [52] under diverse inputs and resource configurations. Results show that the learned models accurately predict fine-grained execution dynamics (such as those shown in Fig. 1) with a median prediction time of at most 1.23 microseconds across all benchmarks. These models generalize to previously unseen inputs, while the conformal prediction framework maintains bounded prediction error with high probability. Our uncertainty-aware dynamic allocation also improves schedulability and response time over existing dynamic allocation methods.

To the best of our knowledge, MPORA is the first framework to formulate multicore resource allocation as an *uncertainty-aware model predictive control problem over learned execution dynamics*. By shifting from static worst-case provisioning to statistically calibrated learning-based control, MPORA enables responsive resource management that improves performance while providing robustness against prediction errors.

2 Related Work

Multicore resource allocation. Early work on multicore resource allocation predominantly relied on static partitioning of shared resources, such as last-level cache and memory bandwidth [61, 60, 44, 45, 55]. These approaches provision resources offline using worst-case assumptions, thus providing strong isolation guarantees but often resulting in poor resource utilization under typical workloads. More recent techniques dynamically allocate resources at runtime [48, 50, 26, 27, 20, 25] to adapt to changing demand. While these approaches improve responsiveness, they often retain static assumptions, such as fixed execution paths, predetermined inputs, or input-independent demand models. MPORA addresses the more practical setting of multi-path programs with runtime inputs unknown *a priori*.

Learning-based workload prediction and phase modeling. Learning techniques have been used extensively to predict execution behavior and improve workload performance. Prior work in this area broadly falls into two categories. The first predicts a job's execution time under a static resource assignment to guide coarse-grained allocations [5, 4, 33, 30]. These methods operate at job granularity and are designed for more relaxed prediction times, making them unsuitable for millisecond-scale allocation decisions. The second category focuses on online [7, 56, 40] or offline [8] program phase detection to inform resource management. However, these models have limitations: some incur inference overheads incompatible with millisecond-scale control decisions [28, 40], others do not model execution behavior under different resource allocations [56], and some assume a fixed execution path [8].

Timing analysis for multi-path, input-dependent programs. Timing analysis is a central topic in real-time systems. Static techniques [14, 36, 31, 10] provide strong guarantees but do not scale with the number of execution paths and resource contexts, making them unsuitable for fine-grained runtime adaptation across arbitrary inputs and resource allocations. In contrast, measurement-based techniques [18, 13, 38, 37] avoid exhaustive analysis but rely on representative input sets, which are inherently incomplete for complex, input-dependent programs [16]. Consequently, offline timing bounds may not reflect runtime behavior.

Feedback control scheduling and resource allocation. Feedback control has long been used to improve the real-time performance by adapting to observed runtime behavior [41, 53]. However, many such solutions are designed to *react* to changes in the system state, targeting high-level performance metrics, such as deadline miss ratios, utilization, or response times [23,

35, 46, 54]. Instead, we take a fine-grained, *intra-job* approach, in which we *plan* resource allocations over a prediction horizon to maximize an optimization objective in upcoming allocation intervals. This introduces distinct challenges which are addressed in this work, including the learning of input/resource/phase-aware task models, uncertainty-quantification under input distribution shifts, and efficient in-kernel inference and optimization.

3 Problem Statement and Optimization Formulation

We first describe the system model and resource allocation problem, then present an offline, clairvoyant optimization formulation that serves as an optimal reference for our proposed online solution. Throughout the paper, we use the notation $\mathbb{N}_0 \coloneqq \{0, 1, 2, \ldots\}$ and $\mathbb{N}^+ \coloneqq \mathbb{N}_0 \setminus \{0\}$. Given $n \in \mathbb{N}^+$, let $[n]_0 \coloneqq \{0, 1, \ldots, n\}$ and $[n] \coloneqq [n]_0 \setminus \{0\}$.

3.1 System Model and Problem Statement

System model. The system consists of a set of $n \in \mathbb{N}^+$ tasks $\Gamma \coloneqq \{\tau_1, \tau_2, \ldots, \tau_n\}$ executing on a multicore platform. Each task τ_i releases a sequence of jobs $\tau_{i,j}$, $j \in \mathbb{N}^+$, with *arbitrary* release times $r_{i,j}$ and arbitrary absolute deadlines $d_{i,j}$, where $r_{i,j} \leq r_{i,j+1}$ and $r_{i,j} < d_{i,j}$. Jobs are scheduled on the cores under a given deterministic scheduling policy $\mathcal{A}$, such as global Earliest Deadline First (EDF) or Fixed-Priority (FP) [15]. As standard in real-time systems, each task is assumed to be a single-threaded, deterministic software program.

We consider a platform with $M \in \mathbb{N}^+$ identical cores and $B \in \mathbb{N}^+$ shared resources (e.g., last-level cache (LLC) and memory bandwidth). Each resource can be divided into a finite number of equal-size partitions using mechanisms such as CAT [32] or LbM [42, 2] for the LLC and MemGuard [63] for memory bandwidth. Each resource $b \in [B]$ has at most $\beta_b^{\max}$ partitions, and each core must be assigned at least $\beta_b^{\min}$ partitions (e.g., $\beta_b^{\max} = 20$ and $\beta_b^{\min} = 2$ on some CAT-enabled platforms). A resource allocation to a core is then represented by a vector with B components, with the b-th component specifying the number of partitions of the b-th resource assigned to the core. Let $\beta^{\min} \coloneqq \left(\beta_1^{\min}, \beta_2^{\min}, \ldots, \beta_B^{\min}\right)$ and $\beta^{\max} \coloneqq (\beta_1^{\max}, \beta_2^{\max}, \ldots, \beta_B^{\max})$, which denote the minimum and maximum allocation per core, respectively. Then, $\mathcal{B} \coloneqq \left\{\beta \in \mathbb{N}_0^B : \beta_b^{\min} \leq \beta_b \leq \beta_b^{\max} \text{ for all } b \in [B]\right\}$ represents the set of all possible resource allocations to a core (or to a job running on a core).

Resource allocation problem. Given a system specified by $(\Gamma, \mathcal{A}, M, B, \mathcal{B})$, our objective is to allocate the B shared resources across the M cores to optimize job response times and total number of instructions retired while meeting deadlines. To adapt to time-varying resource demands, we employ a fine-grained allocation approach that updates resource allocations periodically at fixed intervals of length $\Delta \in \mathbb{N}^+$ time units. Let t_0 be the initial allocation time. At each allocation step $k \in \mathbb{N}_0$, corresponding to time $t_k \coloneqq t_0 + k\Delta$, the allocator selects $\beta_k \coloneqq (\beta_{k,1}, \beta_{k,2}, \ldots, \beta_{k,M}) \in \mathcal{B}^M$, where $\beta_{k,m}$ specifies the allocation assigned on core $m \in [M]$ over the interval $I_k = [t_k, t_k + \Delta)$. For convenience, we denote by $\beta_k^{(i,j)}$ the resources allocated to $\tau_{i,j}$ during I_k (i.e., $\beta_k^{(i,j)} = \beta_{k,m}$) if $\tau_{i,j}$ is scheduled on core m during I_k, and $\beta_k^{(i,j)} = 0$ otherwise. The goal is to determine $\left\{\beta_k\right\}_{k \in \mathbb{N}_0}$, i.e., the distribution of resources to cores at each allocation step, that minimizes total job response time and/or maximizes total number of retired instructions, subject to deadline constraints.

3.2 Job Execution State and Execution Dynamics

On a multicore platform, the execution rate of a job – defined as the number of instructions completed during an interval I_k – depends on its current execution state, including the input-induced execution path, its current progress along this path (i.e, the number of instructions retired at step k), and its sensitivity to allocated resources over I_k. To capture this dependence, we model each job's execution state and its dynamics under a given allocation as follows.

Execution state. For each job $\tau_{i,j}$, let $x_{i,j,k} \in \mathbb{R}^{p_i}$ denote its execution state at allocation step $k \in \mathbb{N}_0$. To track execution progress, we let the first component $x^{(1)}_{i,j,k}$ represent the cumulative number of instructions retired by $\tau_{i,j}$ up to time t_k. Since we consider input-dependent programs, we also include the input value (in the form of a vector) in the execution state of each job. Thus, the dimension $p_i \in \mathbb{N}^+$ is task dependent.

Execution dynamics. For each $i \in [n]$, let $\mu_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$ be an execution rate function that maps a *state–allocation* vector $(x_{i,j,k},\, \beta^{(i,j)}_k)$ to the number of instructions retired by $\tau_{i,j}$ during the interval I_k, where $\tau_{i,j}$ is assigned the resources $\beta^{(i,j)}_k \in \mathcal{B}$ during this interval. We further suppose that the execution state of job $\tau_{i,j}$ satisfies, for each $k \in \mathbb{N}_0$,

$$x_{i,j,k+1} = G_i\left(x_{i,j,k},\, \beta^{(i,j)}_k\right), \tag{1}$$

where $G_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{R}^{p_i}$. Here, G_i is assumed to be induced by μ_i for each $i \in [n]$, and captures the program-specific evolution of a job's execution state. In particular, we have that G_i has its first component given by $x^{(1)}_{i,j,k+1} = x^{(1)}_{i,j,k} + \mu_i(x_{i,j,k}, \beta^{(i,j)}_k)$ for any job $\tau_{i,j}$. Constructing G_i as such enables forward propagation of a job's state across allocation steps, allowing resource allocations chosen at step $k \in \mathbb{N}_0$ to account not only for the immediate interval I_k but also for future intervals $I_{k+\eta} = [t_{k+\eta}, t_{k+\eta} + \Delta)$ for each $\eta \in \mathbb{N}^+$.

3.3 Clairvoyant Optimization Formulation

Given a finite set of jobs $\mathcal{J}$ of taskset Γ, we now present an offline and clairvoyant optimization program to address the resource allocation problem assuming that: 1) the functions μ_i and G_i are known for all $i \in [n]$, and 2) the release time $r_{i,j}$, deadline $d_{i,j}$, and input of each job $\tau_{i,j} \in \mathcal{J}$ are known a priori. We assume the system begins at t_0. The initial execution state of each job $\tau_{i,j} \in \mathcal{J}$ at allocation step $k = 0$ is $x_{i,j,0}$ where $x^{(1)}_{i,j,0} = 0$ (i.e., no instruction has been completed). Since each task in Γ is single-threaded and deterministic, we can determine the terminal execution state $\overline{x}_{i,j}$ of each job $\tau_{i,j}$ under its given input by executing τ_i with the same input. In particular, we define $\overline{x}^{(1)}_{i,j}$ to be the total number of instructions retired by τ_i when executed with the input given for job $\tau_{i,j}$.

Then, the maximum *completion time* and *response time* of each job $\tau_{i,j} \in \mathcal{J}$ are given by

$$C_{i,j} \coloneqq \min\left\{k\Delta : x^{(1)}_{i,j,k} = \overline{x}^{(1)}_{i,j}\right\} + t_0 \quad \text{and} \quad R_{i,j} \coloneqq C_{i,j} - r_{i,j},$$

respectively. In addition, we denote by $\perp$ a job that represents idle execution, with $z \mapsto \mu_\perp(z) = 0$ and $x_{\perp,k} = 0$ for all $k \in \mathbb{N}^+$. Finally, let $\mathcal{S}_\perp \coloneqq (\perp, \ldots, \perp) \in \{\perp\}^M$, representing idle execution at time $t < t_0$.

Let Q_k be the ready queue containing a set of active (released, but unfinished) jobs, and $\mathcal{S}_k$ be the vector of M jobs (potentially including idle jobs $\perp$) that are scheduled on the M cores at allocation step $k \in \mathbb{N}_0$. We assume the job scheduler $\mathcal{A} : (Q_{k+1}, \mathcal{S}_k) \mapsto \mathcal{S}_{k+1}$ is deterministic: given identical ready queues Q_{k+1} at allocation step $k+1$ and identical job schedules $\mathcal{S}_k$ at the previous step k, $\mathcal{A}$ produces the same job schedule $\mathcal{S}_{k+1}$.

Consider a horizon of $N \in \mathbb{N}^+$ allocation steps, where $t_N := t_0 + N\Delta \geq \max_{\tau_{i,j} \in \mathcal{J}} d_{i,j}$. Our goal is to find the allocation β_k for each step $k \in [N-1]_0$ to minimize the total job response times and maximize the total instructions retired, subject to deadline constraints.

Optimization problem. Let $\lambda \in \mathbb{R}_0$. The offline optimal resource allocation is given by:

$$\underset{\{\beta_k\}_{k=0}^{N-1} \subset (\mathcal{B}^M)^N}{\text{maximize}} \quad \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in \mathcal{S}_k} \mu_i(x_{i,j,k},\ \beta_k^{(i,j)}) \ - \ \lambda \sum_{\tau_{i,j} \in \mathcal{J}} R_{i,j} \tag{2}$$

subject to

(C1) Ready queue update: $Q_0 = \{\tau_{i,j} \in \mathcal{J} : r_{i,j} \leq t_0\}$,

$$Q_{k+1} = \left\{\tau_{i,j} \in Q_k : x_{i,j,k+1}^{(1)} < \overline{x}_{i,j}^{(1)}\right\} \cup \left\{\tau_{i,j} \in \mathcal{J} : \left\lceil \frac{r_{i,j} - t_0}{\Delta} \right\rceil = k+1\right\} \quad \forall k \in \mathbb{N}_0$$

(C2) Execution state evolution: $x_{i,j,0} \in \mathbb{R}^{p_i} : \quad x_{i,j,0}^{(1)} = 0$,

$$x_{i,j,k+1} = \begin{cases} x_{i,j,k} & \text{if } \tau_{i,j} \notin S_k \\ G_i\left(x_{i,j,k}, \beta_k^{(i,j)}\right) & \text{otherwise} \end{cases} \quad \forall (i,j) : \tau_{i,j} \in \mathcal{J},\ \forall k \in \mathbb{N}_0$$

(C3) Schedule update: $\mathcal{S}_0 = \mathcal{A}(Q_0, \mathcal{S}_\perp), \quad \mathcal{S}_{k+1} = \mathcal{A}\left(Q_{k+1}, \mathcal{S}_k\right) \quad \forall k \in \mathbb{N}_0$

(C4) Deadline constraint: $d_{i,j} \geq C_{i,j} \quad \forall (i,j) : \tau_{i,j} \in \mathcal{J}$

(C5) Resource constraint: $\sum_{\tau_{i,j} \in \mathcal{S}_k : \tau_{i,j} \neq \perp} \beta_k^{(i,j)} \leq \beta^{\max} \quad \forall k \in \mathbb{N}_0$

Notice that valid partition assignments are encoded in the decision variables of (2). The objective consists of two terms: the first maximizes the total number of instructions retired, while the second minimizes the total job response time. The scaling parameter λ controls the tradeoff between these two objectives, with larger values placing greater emphasis on minimizing response time relative to maximizing instruction retirement.

4 Challenges and Approach Overview

The formulation in (2) assumes that each job's input and execution dynamics are known *a priori*. However, in many real-time systems, job inputs originate from sensors or external environments and thus become available only at *runtime*, typically upon job release. Consequently, precise execution behavior, which is needed for effective resource allocation, cannot generally be determined statically. Below we outline the key challenges of resource allocation under unknown runtime inputs and present the design insights that guide our online solution.

▶ **Challenge 1.** *Inputs of future jobs are unknown at decision time.*

Since inputs of jobs are only available at their releases, an optimal resource allocation computed at time $t \in \mathbb{N}_0$ cannot accurately account for the execution behavior of jobs released after t. As a result, computing a globally optimal allocation is generally infeasible. One way to address this uncertainty is to ignore future allocation steps, i.e., consider only the local allocation at t, or to assume worst-case future demand; however, such approaches are overly conservative and can lead to poor resource utilization.

Insight. Since future job inputs are unavailable at decision time, we must be able to frequently update resource allocation decisions in response to changing resource demand. At the same time, effective resource allocation depends on near-term resource demand and deadline proximity. Therefore, we introduce an online variant of (2) using a finite allocation horizon of $N \in \mathbb{N}_0$ allocation steps. At each allocation step, we compute the optimal resource allocation over the full horizon but apply only the solution for the current step. We then repeat this strategy at each subsequent step. This receding-horizon strategy ensures that decisions are based on currently available information and predicted near-term demand, with regular updates enabling adaptation to phase changes and newly released jobs.

This approach is inspired by model predictive control (MPC), an optimal control technique in which the current state of the system is observed, its trajectory is predicted over a finite window of N steps (the prediction horizon), and a control strategy is computed to minimize some cost functional over that window. Rather than applying all N optimal control actions, MPC commits only to the first $p \leq N$ steps (the control horizon), after which the system state is re-observed and the optimal control problem is solved again. In our setting, the system corresponds to the set of scheduled jobs, and the control action is the distribution of resources to the cores. Our challenge comes from job execution dynamics that depend on runtime inputs, which are unobservable prior to job release, making it infeasible to allocate resources optimally as in (2). Instead, we use predicted execution dynamics to compute the optimal resource allocations over the prediction horizon. Then, since the system state evolved with these predictions cannot reflect jobs or inputs that are not yet observed, we re-solve the optimal control problem at each timestep, applying only the first control action.

▶ **Challenge 2.** *Obtaining an input set that covers all execution paths is often impractical [16].*

The execution path taken by a program – and thus its execution behavior under some resource allocation and at some position in the program – depends on its input. Effective resource allocation with predictions therefore requires input-dependent models.

Insight. We exploit general patterns in execution behavior that persist across inputs, or across certain features of the input, since input vectors can be large in practice. In this case we identify input features that (i) are inexpensive to extract at runtime and (ii) strongly predict remaining execution times and fine-grained execution rates (called "prediction targets" in our online optimization). We then include these features in each job's execution state in lieu of the input vector, allowing predicted execution dynamics to be conditioned on features of the input received at runtime. This enables our models to generalize to unseen inputs.

Prior work has used input features to estimate execution times [33] or parameterize WCETs [29] to improve online performance. However, these approaches mainly target coarse-grained execution metrics for tasks in isolation and assume fixed resource allocation, limiting their applicability. To make holistic resource allocation decisions, we must predict the instruction retirement rate and remaining execution time *at any time t under varying resource allocations*. To this end, we learn a program's *execution pattern*, defined as the temporal evolution of its instruction retirement rate and remaining execution time under different inputs and resource allocations. Like prior work, we treat this prediction problem as a time-series forecasting problem [40]. However, unlike prior work, we require a predictive model that explicitly accounts for program input, phase structure, and resource allocation while delivering microsecond-scale prediction time.

▶ **Challenge 3.** *Predictions are inherently imperfect.*

Our online resource allocation formulation relies on predictive models for each τ_i to evolve the state of each job $\tau_{i,j}$ over the resource allocation horizon and to compare predicted remaining execution time against deadlines. In practice, these models face two sources of error. First, offline training data may not cover all runtime execution paths. Second, even when a path is represented, the models may lack sufficient expressivity to capture the true relationship between execution state, resource allocation, and the prediction targets. These errors can directly affect the safety of the deadline constraint.

Insight. We employ conformal prediction (CP), a distribution-free technique for uncertainty quantification (Sec. 6). Given a predictive model for each τ_i and a held-out calibration set of profiling data, it provides *error bounds* with high probability, provided the calibration and test data are exchangeable. This guarantee is marginal, it holds on average over the joint draw of calibration and test points, rather than conditionally on any particular state-allocation pair. Using these bounds, we can construct uncertainty-aware deadline constraints, tightened to reflect empirical prediction error observed from calibration data.

▶ **Challenge 4.** *Input distributions shift at runtime.*

Standard conformal prediction (CP) provides finite-sample coverage guarantees under the assumption that runtime inputs follow the same distribution as the offline calibration data. In practice, this assumption may not hold, as the distribution of task inputs may evolve over time and change across operating scenarios (e.g., modes) [16].

Insight. Real-time systems often experience input distribution shifts across operating scenarios. Because we assume that software programs are deterministic, the conditional distribution of execution dynamics given the state-allocation vector (which encodes the input vector) is fixed by the software itself. What changes at runtime is the marginal distribution of state-allocation vectors encountered, driven by changes in the input distribution. Therefore, we treat this as *covariate shift*. This structure motivates the use of weighted conformal prediction: for each operating scenario, we learn per-scenario importance weights that reweight the calibration set to reflect the runtime distribution, yielding scenario-specific weighted *error bounds*. Thus, we can adapt uncertainty-aware resource allocation to different input distributions by storing just a single bound per scenario, rather than an entire (separately trained) predictive model.

5 Dynamic Resource Allocation at Runtime

The MPC-based receding-horizon approach allows us to relax the assumption that job inputs, release times, and deadlines are known a priori. However, because the horizon is finite – and typically small in practice to enable efficient online solution[1] – it may not include the completion times and terminal states of all scheduled jobs. As a result, we cannot directly encode the deadline constraint in (C4) or the total response time in the objective (2). Moreover, unlike in the clairvoyance setting, we do not have the ground-truth function μ_i that precisely captures a job's instruction retirement rate under a given resource allocation and, by extension, the execution dynamics G_i of each job $\tau_{i,j}$.

[1] Notice that (2) is a nonlinear integer program with search space $\mathcal{O}((M|\mathcal{B}|)^N)$.

Approach. To address these challenges, using learning-based methods, we approximate the remaining execution time and instruction retirement functions, where for $i \in [n]$:

- $e_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$ maps a job's state-allocation vector to its remaining execution time. Let $x_{i,j,k}$ be the execution state of a job $\tau_{i,j}$ at step $k \in \mathbb{N}_0$. Then $e_i\big(x_{i,j,k}, \beta_k^{(i,j)}\big)$ gives the remaining execution time of $\tau_{i,j}$ assuming it is assigned the resource allocation $\beta_k^{(i,j)}$ from step k until completion (i.e., from time $t_k \coloneqq t_0 + k\Delta$ onwards).
- $\mu_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$ maps a state-allocation vector $\big(x_{i,j,k}, \beta_k^{(i,j)}\big)$ of a job $\tau_{i,j}$ to the number of instructions retired during the interval $I_k \coloneqq [t_k,\, t_k + \Delta)$, assuming that $\tau_{i,j}$ is assigned the resource allocation $\beta_k^{(i,j)}$ during this interval.

For each $i \in [n]$, we denote by $\widehat{e}_i$ and $\widehat{\mu}_i$ the approximations of e_i and μ_i, respectively. We can then obtain an approximation $\widehat{G}_i$ of the true execution dynamics G_i, by replacing the unknown ground-truth μ_i with its approximation $\widehat{\mu}_i$. From these predictions, we estimate job completion and response times, which serve as surrogates for the deadline constraint in (C4) and the objective in (2), respectively.

We next present our online formulation, followed by the construction of $\widehat{e}_i$ and $\widehat{\mu}_i$.

5.1 Online Resource Allocation as an Optimal Control Formulation

Let $t_0 \in \mathbb{N}_0$ denote the *current* resource allocation time (step 0 of the allocation horizon), $N \in \mathbb{N}_0$ the horizon length (in steps), and $k \in \mathbb{N}_0$ the number of allocation steps since t_0. Recall that Q_k and $\mathcal{S}_k$ denote the sets of active jobs and scheduled jobs, respectively. The variable $x_{i,j,k}$ represents the execution state of each job $\tau_{i,j}$ at step k (i.e., time t_k), and we write $\widehat{x}_{i,j,k}$ for the predicted execution state under $\widehat{G}_i$, with $\widehat{x}_{i,j,0} \coloneqq x_{i,j,0}$.

For each $k \in [N-1]_0$, let $\mathcal{J}_k \coloneqq \{\tau_{i,j} \in Q_0 : t_k \le d_{i,j} < t_{k+1}\}$ be the set of jobs in Q_0 with deadlines in the interval I_k. Let $\mathcal{J}_N \coloneqq Q_0 \setminus \bigcup_{k\in[N-1]_0} \mathcal{J}_k$ be the set of jobs in Q_0 with deadlines at or beyond the allocation horizon (i.e., $d_{i,j} \ge t_N$).

Online optimization formulation. At the current allocation time t_0, the active job set Q_0, scheduled job set $\mathcal{S}_0$, and execution states $x_{i,j,0}$ are sampled from the running system. We set $\widehat{x}_{i,j,0} = x_{i,j,0}$. The online version of (2) is

$$\underset{\{\beta_k\}_{k=0}^{N-1} \subset (\mathcal{B}^M)^N}{\text{maximize}} \quad \sum_{k=0}^{N-1} \sum_{\tau_{i,j}\in\mathcal{S}_k} \widehat{\mu}_i(\widehat{x}_{i,j,k}, \beta_k^{(i,j)}) \;-\; \lambda \sum_{\tau_{i,j}\in Q_0} \widehat{R}_{i,j}. \tag{3}$$

subject to

(D1) Ready queue update: $Q_{k+1} = \big\{\tau_{i,j} \in Q_k : \widehat{e}_i(\widehat{x}_{i,j,k},\ \beta^{\min}) > 0\big\} \quad \forall k \in [N-1]_0$

(D2) Execution state:

$$\widehat{x}_{i,j,k+1} = \begin{cases} \widehat{x}_{i,j,k} & \text{if } \tau_{i,j} \notin \mathcal{S}_k \\ \widehat{G}_i\left(\widehat{x}_{i,j,k}, \beta_k^{(i,j)}\right) & \text{otherwise} \end{cases} \qquad \forall (i,j) : \tau_{i,j} \in Q_0, \quad \forall k \in [N-1]_0$$

(D3) Schedule update: $\mathcal{S}_{k+1} = \mathcal{A}(Q_{k+1}, \mathcal{S}_k) \quad \forall k \in [N-1]_0$

(D4) Deadline constraint:

$$\begin{aligned} \widehat{e}_i(\widehat{x}_{i,j,k}, \beta^{\min}) \;+\; t_0 + k\Delta \;&\le\; d_{i,j} \qquad \forall k \in [N-1]_0, \quad \forall (i,j) : \tau_{i,j} \in \mathcal{J}_k \\ \widehat{e}_i(\widehat{x}_{i,j,N}, \beta_{N-1}^{(i,j)}) \;+\; t_0 + N\Delta \;&\le\; d_{i,j} \qquad\qquad\quad \forall (i,j) : \tau_{i,j} \in \mathcal{J}_N \end{aligned}$$

(D5) Resource constraint: $\sum_{\tau_{i,j}\in\mathcal{S}_k : \tau_{i,j} \neq \perp} \beta_k^{(i,j)} \le \beta^{\max} \qquad \forall k \in [N-1]_0$

(D6) Response time:

$$k_{i,j} \;=\; \max\{k \in [N-1]_0 \;:\; \widehat{e}_i\big(\widehat{x}_{i,j,k}, \beta_k^{(i,j)}\big) > 0\} \qquad \forall (i,j) : \tau_{i,j} \in Q_0$$
$$\widehat{R}_{i,j} \;=\; \widehat{e}_i\big(\widehat{x}_{i,j,k_{i,j}}, \beta_{k_{i,j}}^{(i,j)}\big) + t_0 + k_{i,j}\Delta - r_{i,j} \qquad \forall (i,j) : \tau_{i,j} \in Q_0$$

Configuration of Δ and N. The step size Δ and horizon length N introduce a tradeoff. The complexity of (3) grows exponentially with N. To keep overhead low (e.g., below $\epsilon(\%)$), solving (3) must take less than $\epsilon(\%)$ of the allocation interval Δ multiplied by the number of cores M, which effectively requires Δ to grow exponentially with N. Since small Δ is essential for responsiveness in dynamic execution environments, N must remain small in practice. Larger horizons, however, allow less conservative decisions: allocations within the horizon are optimized at each step, while execution beyond the horizon is aggregated under a single static allocation to estimate remaining execution time (D4). Finally, the choice of Δ affects the accuracy of $\widehat{e}_i$ and $\widehat{\mu}_i$ independently of N.

5.2 Learning Multi-Path Program Behavior

Our online formulation relies on functions $\widehat{\mu}_i$ and $\widehat{e}_i$, $i \in [n]$, to estimate the instructions retired and remaining execution time of each job within the allocation horizon. Deriving such functions from first principles is generally intractable due to the complex interactions among a program's input, phase structure, and allocated resources [16]. Therefore, we construct $\widehat{\mu}_i$ and $\widehat{e}_i$ via learning-based methods.

Importantly, the model class must be sufficiently expressive to capture these complex interactions while maintaining microsecond-scale prediction speeds. Towards this, we employ Gradient Boosting Regression Trees (GBRT) as our predictive models. GBRTs provide established generalization guarantees [22] and enable highly efficient inference, since each prediction consists of evaluating a fixed number of shallow decision trees using only simple comparison operations and additions. This makes them particularly well-suited for kernel-level deployment. In fact, the GBRT model – specifically the XGBoost implementation [12] that we use – has been shown to outperform even sophisticated deep learning models at time-series forecasting [21] on structured data such as ours (recall our data consists of discrete measurements of retired instructions conditioned on input vectors and resource allocation).

For each $i \in [n]$, let $\mathcal{H}_i$ be the set of admissible regression trees mapping from $\mathbb{R}^{p_i} \times \mathcal{B}$ to $\mathbb{R}$. Then, for each task $\tau_i \in \Gamma$, our objective is to learn the ensembles of regression trees $\widehat{\mu}_i$ and $\widehat{e}_i$ that best represent the relationship between execution state, resource allocation, and the corresponding prediction target. To accomplish this, we first collect offline execution profiles for each task τ_i. This process involves repeatedly executing τ_i over many runs under a diverse set of inputs and resource allocations and, in each run, measuring its state every Δ time units. Thus, for each $i \in [n]$, we define the dataset

$$\mathcal{D}_i \coloneqq \big\{\big(z_{i,j}, \mu_i(z_{i,j})\big) + \varepsilon_j\big\}_{j=1}^{\gamma_i} \subset \mathbb{R}^{p_i} \times \mathcal{B} \times \mathbb{N}_0 \tag{4}$$

where $\gamma_i \in \mathbb{N}^+$ denotes the total number of measurements collected for task τ_i, $z_{i,j} \coloneqq (x_{i,j}, \beta_{i,j}) \in \mathbb{R}^{p_i} \times \mathcal{B}$ is the state-allocation vector associated with the j-th measurement, and $\varepsilon_j \in \mathbb{N}$ models measurement noise. Using standard tools from supervised regression, such as *empirical risk minimization* (see [57, Sec. 8.4.3] for more details), we select a $\widehat{\mu}_i \in \mathcal{H}_i$ based on the dataset $\mathcal{D}_i$ for each τ_i. We repeat the same procedure for $\widehat{e}_i \in \mathcal{H}_i$ by replacing μ_i with e_i for each $i \in [n]$ in (4).

In the next section, we discuss how we quantify the inherent prediction error of these learned models, to formulate robust deadline constraints.

6 Uncertainty-Aware Dynamic Resource Allocation

This section describes how we incorporate data-driven uncertain estimates into dynamic resource allocation to provide probabilistic guarantees even when runtime behavior shifts. We begin by introducing conformal prediction, a lightweight, distribution-free method for quantifying prediction uncertainty. We then show how to enhance its robustness to unknown distribution shift through sample reweighting. Finally, we demonstrate how these uncertainty bounds can be integrated into our online optimization formulation.

6.1 Distribution-Free Uncertainty Quantification Background

We build on *conformal prediction* (CP) [58], a state-of-the-art, lightweight, general framework for distribution-free uncertainty quantification. First introduced in [58], advances in CP trace back to the development of *inductive conformal prediction* [43], later reformulated as *split conformal prediction* (SCP) in [34]. At a high level, CP quantifies how wrong a predictive model tends to be by calibrating to the empirical distribution of past prediction errors, independent of their underlying source, thus, providing finite-sample guarantees without requiring any assumptions on the underlying data distribution. Without loss of generality, we focus on uncertainty quantification for the predictions of $\widehat{\mu}_i$ for each task τ_i, $i \in [n]$, as the results extend straightforwardly to $\widehat{e}_i$.

Objective. For each task τ_i, consider a dataset $\mathcal{D}_i$ as defined in (4), consisting of $\gamma_i \in \mathbb{R}_{>0}$ measurements. The dataset comprises measured state-allocation vectors $\{z_{i,j}\}_{j=1}^{\gamma_i}$ and corresponding prediction targets (e.g., number of instructions retired) $\{y_{i,j}\}_{j=1}^{\gamma_i}$, where each $z_{i,j}$ consists of an execution state of a job of τ_i and a corresponding resource allocation. Given a desired miscoverage rate $\delta_i \in (0,1)$, our goal is to construct a set-valued map $\mathcal{E}_{\delta_i} : \mathbb{R}^{p_i} \times \mathcal{B} \rightrightarrows \mathbb{R}$ that assigns a subset of $\mathbb{R}$ to each $z \in \mathbb{R}^{p_i} \times \mathcal{B}$ such that, for a new observed state-allocation vector z_{i,γ_i+1} of some execution of τ_i, its prediction target y_{i,γ_i+1} lies within $\mathcal{E}_{\delta_i}(z_{i,\gamma_i+1})$ with probability at least $1-\delta_i$.

Weighted split conformal prediction. To achieve the above goal, we split $\mathcal{D}_i$ into two disjoint sets: a training set $\mathcal{D}_i^{\text{train}}$ of size γ_i^{train} and a calibration set $\mathcal{D}_i^{\text{cal}}$ of size γ_i^{cal}, with $\gamma_i^{\text{train}} + \gamma_i^{\text{cal}} = \gamma_i$. The predictive model $\widehat{\mu}_i$ is obtained as discussed in Section 5.2 using $\mathcal{D}_i^{\text{train}}$. We then evaluate its *conformity* – a quantitative measure of how well its predictions match the observed values in the calibration set – on the calibration samples in $\mathcal{D}_i^{\text{cal}}$ via the *nonconformity scores* defined, for each $j \in \left[\gamma_i^{\text{cal}} + 1\right]$, as $S_{i,j} := \left|\widehat{\mu}_i(z_{i,j}) - y_{i,j}\right|$, where smaller scores correspond to higher predictive accuracy. If the joint distribution of the nonconformity scores $\left\{S_{i,j}\right\}_{j=1}^{\gamma_i^{\text{cal}}+1}$ has the same distribution as $\left\{S_{i,\sigma(j)}\right\}_{j=1}^{\gamma_i^{\text{cal}}+1}$, under any permutation σ of the indices $1, 2, \ldots, \gamma_i^{\text{cal}}+1$ (this is known as *exchangeability*), then results in the literature, in particular [34, Thm. 2.2], provide tools to construct $\mathcal{E}_{\delta_i}$ given a miscoverage rate $\delta_i \in (0,1)$. In practice, however, this assumption does not generally hold. For real-time systems, in particular, the distribution of runtime task inputs may drift over time and is unknown a priori. Further, since resource allocations are temporally coupled – insufficient resources at one timestep can increase future demand – permuting samples across time can break causal dependencies, invalidating exchangeability.

To relax this assumption, the authors in [9] propose *robust conformal prediction*, which provides valid guarantees without being too conservative when the distribution shift is known and small; however, large distribution shifts can result in conservative prediction regions[2].

[2] The reader is referred to [1, 24, 62] for further related and recent work on robust conformal prediction.

Since the magnitude and structure of distribution shifts are unknown in our setting, we instead adopt weighted conformal thresholds [6] to improve robustness to distribution drift. Precisely, for each $i \in [n]$ and each $j \in \left[\gamma_i^{\text{cal}}\right]$, let $w_{i,j} \in [0,1]$ and define

$$\tilde{w}_{i,j} \coloneqq \frac{w_{i,j}}{w_{i,1} + w_{i,2} + \cdots + w_{i,\gamma_i^{\text{cal}}} + 1}, \qquad \tilde{w}_{i,\gamma_i^{\text{cal}}+1} \coloneqq \frac{1}{w_{i,1} + w_{i,2} + \cdots + w_{i,\gamma_i^{\text{cal}}} + 1}. \tag{5}$$

For each $\delta_i \in (0,1)$, the *weighted conformal threshold* is given by

$$\widehat{q}_{\delta_i}^{\mu} \coloneqq \inf\left\{ q \in \mathbb{R} : \sum_{j=1}^{\gamma_i^{\text{cal}}} \tilde{w}_{i,j}\, \mathbb{1}_{\{S_{i,j} \le q\}} \ge 1 - \delta_i \right\} \tag{6}$$

with the usual convention that $\inf \emptyset = +\infty$, which happens if $\delta_i < \tilde{w}_{i,\gamma_i^{\text{cal}}+1}$. Then, by [6, Thm. 2], we have that

$$\mathbb{P}\Big\{ y_{\gamma_i^{\text{cal}}+1} \in \widehat{\mathcal{E}}_{\delta_i}\left(z_{\gamma_i^{\text{cal}}+1}\right) \Big\} \ge 1 - \delta_i - \sum_{j=1}^{\gamma_i^{\text{cal}}} \tilde{w}_{i,j}\, \mathrm{d_{TV}}\left(S_{i,j}^{\text{cal}}, \overline{S}_{i,j}\right) \tag{7}$$

where the metric $\mathrm{d_{TV}}$ denotes the total variation distance between two probability distributions, $S_{i,j}^{\text{cal}}$ is the set of nonconformity scores for $\left\{(z_{i,j}, y_{i,j})\right\}_{j=1}^{\gamma_i^{\text{cal}}+1}$, $\overline{S}_{i,j}$ is the set of nonconformity scores when $\left(z_{i,\gamma_i^{\text{cal}}+1}, y_{i,\gamma_i^{\text{cal}}+1}\right)$ is swapped with the j-th sample in $S_{i,j}^{\text{cal}}$, and

$$\widehat{\mathcal{E}}_{\delta_i}(z) \coloneqq \left\{ y \in \mathbb{R} : |\widehat{\mu}_i(z) - y| \le \widehat{q}_{\delta_i}^{\mu} \right\} \qquad \forall z \in \mathbb{R}^{p_i} \times \mathcal{B}. \tag{8}$$

Notice that the coverage guarantee decreases with distribution shift since the probability is bounded by $1 - \delta_i$ minus a term that quantifies the dissimilarity between the weighted calibration samples and runtime behavior, measured via a total variation distance $\mathrm{d_{TV}}$. Also note that the set-valued map $\widehat{\mathcal{E}}_{\delta_i}$ is independent of the unknown distribution shift, enabling it to be computed in closed form.

6.2 Calibrating the Predictor

Our objective in this section is to *learn* a weighting rule that emphasizes calibration samples most representative of runtime behavior. Intuitively, if the weights concentrate on calibration points most resembling runtime state-allocation vectors, the weighted empirical quantile better reflects the effective test-time distribution of the nonconformity scores. Consequently, the achieved coverage is closer to the target $1 - \delta_i$, which is the best-case scenario under exchangeability, while avoiding overly conservative prediction sets.

One way to construct a calibration set representative of runtime behavior is exploiting the fact that many real-time systems operate in a finite set of operating scenarios, or modes, corresponding to changes in the internal or external state. For each scenario, we can collect a representative set of inputs for each task and form a per-scenario calibration set by profiling the task with these inputs. We then learn a separate weighting rule for each calibration set to maximize the coverage guarantee, enabling the uncertainty quantification to adapt to distribution changes at runtime.

A natural strategy for learning the weights for τ_i is to treat the weights $\{w_{ij}\}_{j=1}^{\gamma_i^{\text{cal}}}$ as decision variables and directly optimize them (e.g., via Bayesian optimization or genetic algorithms) to maximize empirical coverage over a held-out dataset that resembles runtime behavior. However, this strategy has two main drawbacks. First, optimizing a high-dimensional weight vector in this way is computationally expensive and scales poorly with the number of

calibration samples, making gradient-free methods expensive past moderate γ_i^{cal}. Second, empirical coverage is not a smooth function, which encourages degenerate solutions that concentrate weight on a small subset of calibration points; such solutions overfit the held-out set, produce unstable thresholds that depend on a few samples, and may either over- or under-cover at runtime depending on which samples receive weight. These challenges motivate *parameterizing the weights* as a function of the state-allocation vector and learning a model that can generalize over the operating scenario.

More precisely, for each $i \in [n]$, let the calibration dataset $\mathcal{D}_i^{\mathrm{cal}}$ be fixed offline for each scenario of operation, and let the weights be computed once from this dataset. To obtain a stable, scalable, and structured weighting scheme, we parameterize the weights with a low-dimensional model (e.g., a neural network), parametrized by a vector $\theta \in \mathbb{R}^{n_{\theta_i}}$ that defines a map $z \mapsto \widehat{w}_{i,j}^{\theta}(z) \in [0,1]$ over calibration features. After learning θ offline, we evaluate this map only on $\{z_{i,j}\}_{j=1}^{\gamma_i^{\mathrm{cal}}}$ to obtain a fixed set of weights, which is then used to compute $\widehat{q}_{\delta_i}$. At runtime, only $\widehat{q}_{\delta_i}$ must be stored for each operating scenario, resulting in a negligible memory footprint.

To formalize how θ is computed, suppose a miscoverage rate $\delta_i \in (0,1)$ and a dataset $\mathcal{D}_i^{\mathrm{wt}} \coloneqq \{S_{i,j}\}_{j=1}^{\gamma_i^{\mathrm{wt}}}$ are given for each scenario (as discussed above), and let q_i^* be the $(1-\delta_i)^{\mathrm{th}}$-quantile of $\mathcal{D}_i^{\mathrm{wt}}$. Next, for each $\delta \in (0,1)$, we define the *quantile loss* $\rho_{1-\delta} : \mathbb{R} \to \mathbb{R}_{\geq 0}$ as

$$\rho_{1-\delta}(\alpha) \coloneqq \begin{cases} (1-\delta)\alpha & \text{if } \alpha \geq 0 \\ -\delta\alpha & \text{if } \alpha < 0 \end{cases} \qquad \forall \alpha \in \mathbb{R}. \tag{9}$$

Thus, for each $i \in [n]$, we propose to solve

$$\theta_i^* \in \operatorname*{arg\,inf}_{\theta \in \mathbb{R}^{n_{\theta_i}}} \frac{1}{\gamma_i^{\mathrm{cal}}} \sum_{j=1}^{\gamma_i^{\mathrm{cal}}} \tilde{w}_{i,j}^{\theta}(z_{i,j}) \Big(\rho_{1-\delta_i}(S_{ij} - q_i^*) + \psi_1 \log \tilde{w}_{i,j}^{\theta}(z_{i,j}) \Big) + \psi_2 |\theta|^2, \tag{10}$$

where $\psi_1, \psi_2 > 0$, and $\tilde{w}_{i,j}^{\theta}$ is computed as in (5) using $\widehat{w}_{i,j}^{\theta}$. Following [47, Thm. 1.9], it can be shown that the right-hand side of (10) is nonempty and compact[3], and thus (10) has at least one solution.

The objective in (10) can be interpreted as follows: For each $i \in [n]$, the threshold q_i^* is fixed offline as the empirical $(1-\delta_i)^{\mathrm{th}}$-quantile of the nonconformity scores in $\mathcal{D}_i^{\mathrm{wt}}$. The quantile loss encourages the learned weights to concentrate on calibration samples whose scores resemble q_i^*, so that the threshold computed via (6) reflects the distribution of the scenario of operation represented by $\mathcal{D}_i^{\mathrm{wt}}$. The entropy term, $\tilde{w}_{i,j}^{\theta}(z_{i,j}) \log \tilde{w}_{i,j}^{\theta}(z_{i,j})$, acts as a maximum-entropy regularizer, penalizing solutions that place weight on a small subset of calibration samples and thereby preventing degenerate minimizers. Finally, the regularization term $\psi_2|\theta|^2$ ensures existence of a minimizer.

6.3 Conformal Online Resource Allocation

We now show how the weighted conformal prediction bounds derived in the previous sections can be integrated into the online optimization formulation developed in Section 5.1 to provide deadline constraints that account for prediction error. To develop this argument, we will suppose that, for each $i \in [n]$, the map $z \mapsto \big(\widehat{e}_i(z), \widehat{\mu}_i(z)\big)$ is Lipschitz continuous; that is, there exists $L \in [0, \infty)$ such that, for all $z, z' \in \mathbb{R}^{p_i} \times \mathcal{B}$, it holds that $\big|(\widehat{e}_i(z), \widehat{\mu}_i(z)) - (\widehat{e}_i(z'), \widehat{\mu}_i(z'))\big| \leq L|z - z'|$.

[3] This follows from the fact that the objective in (10) is (1) proper since $\operatorname{rge} \rho_{1-\delta} \subset [0, \infty)$ and $p \log p \geq -1/e$ for all $p \in (0,1]$ together imply that the domain of the objective is a nonempty set on which it is finite, (2) continuous in θ, and (3) radially unbounded.

▶ Remark 1 (Lipschitz continuous approximations of GBRTs). Since a GBRT is an ensemble of regression trees (hRTs), it can be shown that each hRT admits a smooth approximation that converges pointwise almost everywhere. Thus, we can exploit this fact to estimate Lipschitz constants of sRTs on compact sets and, therefore, for any finite ensemble of them.

Notice that, for each $i \in [n]$, given a miscoverage rate $\delta_i \in (0,1)$, by (7) and (8), there exist $\widehat{\delta}_i^e \geq \delta_i$, $\widehat{\delta}_i^\mu \geq \delta_i$, and $\widehat{q}_{\delta_i}^e, \widehat{q}_{\delta_i}^\mu \in (0,\infty)$ such that

$$\mathbb{P}\left\{\star_i(z) \leq \widehat{\star}_i(z) + \widehat{q}_{\delta_i}^\star\right\} \geq 1 - \widehat{\delta}_i^\star \qquad \forall \star \in \{e, \mu\} \tag{11}$$

where the probability is taken jointly over the calibration set and a test pair $(z, \star_i(z))$.

Error propagation of the state over the allocation horizon. From (D2), notice that, for each $i \in [n]$, the predicted execution state $\widehat{x}_{i,j,k}$ is computed by rolling out $\widehat{G}_i$ over k steps from the initial condition $x_{i,j,0}$. In practice, the map $z \mapsto \widehat{\mu}_i(z)$ is obtained via empirical risk minimization over the dataset $\mathcal{D}_i$ (as discussed in Section 5.2), and as such it carries irreducible prediction error. Namely, the learned model may not have seen all state-allocation vectors and their corresponding prediction target, and even for paths that are represented in $\mathcal{D}_i$, the regression tree ensemble has finite expressivity. Since $\widehat{G}_i$ is defined through $\widehat{\mu}_i$, the uncertainty thus propagates along allocation steps, and quantifying it explicitly is essential to ensure the validity of the optimization constraints.

To address this issue, we present the following result.

▶ **Lemma 2.** *For each $i \in [n]$, consider $\overline{G}_i : \mathbb{R}^{p_i} \times \mathbb{N}_0 \times \mathcal{B} \to \mathbb{R}^{p_i}$, $\mu_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$, and $\widehat{\mu}_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$. Suppose that, for each $\beta \in \mathcal{B}$, the map $(x,y) \mapsto \overline{G}_i(x,y,\beta)$ is Lipschitz with constant $L_{\overline{G}_i} \in [0,\infty)$, and that $\widehat{\mu}_i$ is also Lipschitz with constant $L_{\mu_i} \in [0,\infty)$. Given $\delta_i \in (0,1)$, suppose that (11) holds with miscoverage rate $\widehat{\delta}_i^\mu \geq \delta_i$. Let $x_{i,j,k}$ and $\widehat{x}_{i,j,k}$ be the true and predicted execution states of each job $\tau_{i,j}$ at allocation step $k \in \mathbb{N}_0$, obtained by rolling out $G_i = \overline{G}_i \circ (\mathrm{id}, \mu_i, \mathrm{id})$ and $\widehat{G}_i = \overline{G}_i \circ (\mathrm{id}, \widehat{\mu}_i, \mathrm{id})$, respectively, from a given $x_{i,j,0}$. Then, for each $k \in \mathbb{N}_0$, it follows that*

$$\mathbb{P}\left\{|x_{i,j,k} - \widehat{x}_{i,j,k}| \leq \rho_k\right\} \geq 1 - k\widehat{\delta}_i^\mu, \tag{12}$$

where

$$\rho_k \coloneqq \begin{cases} L_{\overline{G}_i}\widehat{q}_{\delta_i}^\mu \dfrac{\left(L_{\overline{G}_i}(1+L_{\mu_i})\right)^k - 1}{L_{\overline{G}_i}(1+L_{\mu_i}) - 1} & \text{if } L_{\overline{G}_i}(1+L_{\mu_i}) \neq 1 \\ k L_{\overline{G}_i}\widehat{q}_{\delta_i}^\mu & \text{if } L_{\overline{G}_i}(1+L_{\mu_i}) = 1 \end{cases} \qquad \forall k \in \mathbb{N}_0. \tag{13}$$

Proof. For ease of notation, let $z_{i,j,k}$ denote the state-allocation vector of job $\tau_{i,j}$ at allocation step $k \in \mathbb{N}_0$. By the union bound, together with (11), it follows that

$$\begin{aligned} \mathbb{P}\left\{\bigcap_{\ell=0}^{k-1} \left|\mu_i\left(z_{i,j,\ell}\right) - \widehat{\mu}_i\left(z_{i,j,\ell}\right)\right| \leq \widehat{q}_{\delta_i}^\mu\right\} &\geq 1 - \sum_{\ell=0}^{k-1} \mathbb{P}\left\{\left|\mu_i\left(z_{i,j,\ell}\right) - \widehat{\mu}_i\left(z_{i,j,\ell}\right)\right| > \widehat{q}_{\delta_i}^\mu\right\} \\ &\geq 1 - k\widehat{\delta}_i^\mu. \end{aligned}$$

We now show by induction that if, for all $\ell \in [k-1]_0$, we have that $\left|\mu_i\left(z_{i,j,\ell}\right) - \widehat{\mu}_i\left(z_{i,j,\ell}\right)\right| \leq \widehat{q}_{\delta_i}^\mu$, then $|x_{i,j,k} - \widehat{x}_{i,j,k}| \leq \rho_k$. For the base case, take $k = 0$. By assumption $x_{i,j,0} = \widehat{x}_{i,j,0}$, thus $|x_{i,j,0} - \widehat{x}_{i,j,0}| = 0 = \rho_0$. Next, pick any $k \in \mathbb{N}^+$ and suppose that $|x_{i,j,k} - \widehat{x}_{i,j,k}| \leq \rho_k$ holds.

Then,

$$\begin{aligned}
|x_{i,j,k} - \widehat{x}_{i,j,k}| &= \left|\overline{G}_i\left(x_{i,j,k}, \mu_i(z_{i,j,k}), \beta_k^{(i,j)}\right) - \overline{G}_i\left(\widehat{x}_{i,j,k}, \widehat{\mu}_i(\widehat{z}_{i,j,k}), \beta_k^{(i,j)}\right)\right| \\
&\leq L_{\overline{G}_i}\left(\left|x_{i,j,k} - \widehat{x}_{i,j,k}\right| + \left|\mu_i(z_{i,j,k}) - \widehat{\mu}_i(\widehat{z}_{i,j,k})\right|\right) \\
&\leq L_{\overline{G}_i}\left(\left|x_{i,j,k} - \widehat{x}_{i,j,k}\right| + \left|\mu_i(z_{i,j,k}) - \widehat{\mu}_i(z_{i,j,k})\right| + \left|\widehat{\mu}_i(z_{i,j,k}) - \widehat{\mu}_i(\widehat{z}_{i,j,k})\right|\right) \\
&\leq L_{\overline{G}_i}\left(\widehat{q}^{\mu}_{\delta_i} + \rho_k(1 + L_{\mu_i})\right) \\
&= \rho_{k+1}.
\end{aligned}$$

The proof is completed by noticing that, with $\rho_0 = 0$, (13) is the solution to the recursion $\rho_{k+1} = L_{\overline{G}_i}\left(\widehat{q}^{\mu}_{\delta_i} + \rho_k(1 + L_{\mu_i})\right)$ for all $k \in \mathbb{N}_0$. ◀

Uncertainty-aware deadline constraint. Recall that the deadline constraint (D4) requires the predicted remaining execution time to satisfy the deadline of each job $\tau_{i,j}$. For each $i \in [n]$, like $\widehat{\mu}_i$, the map $z \mapsto \widehat{e}_i(z)$ is obtained via empirical risk minimization with finite amount of data and, therefore, carries irreducible sources of prediction error. To address this, we formulate an uncertainty-aware deadline constraint using the following result, which gives a probabilistic upper bound on the predictions of $z \mapsto \widehat{e}_i(z)$.

▶ **Lemma 3.** *For each $i \in [n]$, consider the functions $\overline{G}_i : \mathbb{R}^{p_i} \times \mathbb{N}_0 \times \mathcal{B}$, $e_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$ and $\widehat{e}_i : \mathbb{R}^{p_i} \times \mathcal{B} \to \mathbb{N}_0$. Suppose that $\widehat{e}_i$ is Lipschitz with constant $L_{e_i} \in [0, \infty)$ and that the assumptions in Lemma 2 hold. Then, for each $k \in \mathbb{N}_0$, each $\beta \in \mathcal{B}$, and each $\delta_i \in (0, 1)$, it follows that*

$$\mathbb{P}\left\{e_i\big(x_{i,j,k}, \beta\big) \leq \widehat{e}_i\big(\widehat{x}_{i,j,k}, \beta\big) + \widehat{q}^{e}_{\delta_i} + L_{e_i}\rho_k\right\} \geq 1 - \widehat{\delta}^{e}_i - k\widehat{\delta}^{\mu}_i, \tag{14}$$

where $\widehat{q}^{e}_{\delta_i} \in (0, \infty)$, $\widehat{\delta}^{e}_i \geq \delta_i$, and $\widehat{\delta}^{\mu}_i \geq \delta_i$ come from (11), and ρ_k is defined in (13).

Proof. Pick any $k \in \mathbb{N}_0$ and any $\beta \in \mathcal{B}$. Using the triangle inequality, notice that

$$\begin{aligned}
e_i(x_{i,j,k}, \beta) - \widehat{e}_i(\widehat{x}_{i,j,k}, \beta) &\leq \left|e_i(x_{i,j,k}, \beta) - \widehat{e}_i(x_{i,j,k}, \beta)\right| + \left|\widehat{e}_i(x_{i,j,k}, \beta) - \widehat{e}_i(\widehat{x}_{i,j,k}, \beta)\right| \\
&\leq \widehat{q}^{e}_{\delta_i} + L_{e_i}\left|x_{i,j,k} - \widehat{x}_{i,j,k}\right| \\
&\leq q^{e}_{\delta_i} + L_{e_i}\rho_k.
\end{aligned}$$

Since the first term in the previous inequality holds with probability at least $1 - \widehat{\delta}^{e}_i$ and the second one with probability at least $1 - k\widehat{\delta}^{\mu}_i$, the proof is readily completed using a union bound argument. ◀

By a straightforward application of Lemma 3, we can replace the constraint (D4) with

($\mathrm{D_p}$4) Uncertainty-aware deadline constraint

$$\widehat{e}_i(\widehat{x}_{i,j,k}, \bar{\beta}) + \widehat{q}^{e}_{\delta_i} + L_{e_i}\rho_k + t_0 + k\Delta \leq d_{i,j} \qquad \forall k \in [N]_0, \quad \forall (i,j) : \tau_{i,j} \in \mathcal{J}_k,$$

where $\bar{\beta} = \beta^{\min}$ if $k < N$ and $\bar{\beta} = \beta^{(i,j)}_{N-1}$ if $k = N$.

It is important to mention that the constraint in ($\mathrm{D_p}$4) is not probabilistic itself, it is a deterministic constraint on the predicted remaining execution time. However, its implication for (D4) holds with probability at least $1 - \widehat{\delta}^{e}_i - k\widehat{\delta}^{\mu}_i$. In addition, notice that the deadline constraint is now tightened by $\widehat{q}^{e}_{\delta_i} + \rho_k$, which accounts for two distinct sources of error: the prediction error of $z \mapsto \widehat{e}_i(z)$ at the allocation step k, quantified by the conformal threshold $\widehat{q}^{e}_{\delta_i}$, and the error introduced by evaluating $z \mapsto \widehat{e}_i(z)$ at the predicted state $\widehat{x}_{i,j,k}$ rather than the *true* state $x_{i,j,k}$.

▶ Remark 4. The weighted conformal bounds in (11) are marginally valid: the probability is taken over the joint distribution of the calibration set and a test state-allocation pair from the runtime distribution. However, following [11], we apply these bounds in Lemma 2 and in Lemma 3 heuristically as per-step uncertainty estimates along the trajectory, even though this is generated deterministically by $\widehat{G}_i$. Achieving formally per-trajectory coverage would require additional machinery, see [39], which we leave to future work.

7 Predictive Model Evaluation

To empirically evaluate the accuracy of models μ_i and e_i for each $\tau_i \in \Gamma$, we used programs from the SPEC benchmark suite [52], as it has been widely adopted in multicore resource-allocation and phase detection research [40, 56]. We begin with our experimental methodology.

7.1 Input Generation

For each SPEC benchmark, we constructed an input set containing 200 distinct inputs that capture a broad range of execution behaviors. Each input set was designed to be both *diverse*, covering typical and worst-case behaviors, and *representative*, reflecting realistic workloads for the target application domain. When benchmark-specific datasets are available, we used them directly; otherwise, we generated inputs randomly according to each benchmark's input specification. All inputs were subject to only a single global constraint: a maximum input size. Such bounds are standard in real-time systems to enable analyzability, e.g., worst-case execution-time (WCET) analysis. We describe the input-generation procedure for each benchmark below. Unless otherwise noted, all benchmarks are implemented in C/C++. The resulting inputs yield execution times ranging from approximately 20 ms to over 4 s.

- *mcf* **(route planning)** solves a minimum-cost flow formulation for vehicle scheduling using a network simplex algorithm. Each input specifies N timetabled trips (nodes) and M feasible links (edges) between them. Each trip $u \in [N]$ is characterized by a start time $t_{u,s} \in \mathbb{N}$ and a finish time $t_{u,f} \in \mathbb{N}$. A directed link (u, v) represents a feasible vehicle transition from trip u to trip v (i.e., $t_{u,f} \leq t_{v,s}$), with an associated cost $c_{(u,v)} \in \mathbb{N}$. We generated inputs by sampling the number of trips $N \sim \text{Uniform}(10, 500)$ and assigning each trip random start and finish times within a one-day horizon. We generated feasible links with probability $p \sim \text{Uniform}(0.01, 0.5)$, each with link cost $c_{(u,v)} \sim \text{Uniform}(1, 100)$.
- *omnetpp* **(discrete event simulation)** simulates a 10 Gb Ethernet network using discrete-event simulation and takes network configuration files as inputs. We generated inputs by varying network parameters uniformly within realistic ranges, including the number of hosts connected to a switch, hub, and bus in each LAN from Uniform$(1, 10)$, request and response sizes (50–1400 bytes), and inter-request delays (0–1000 ms). The simulation time was capped at 0.001 s, yielding execution times of up to 2 s.
- *xalancbmk* **(data conversion)** performs XML-to-HTML transformation using an XSL stylesheet. Its execution behavior is primarily determined by the size and structural complexity of the input XML document. We generated inputs using the open-source `xmlgen` tool from the XMark project [49], with target sizes uniformly sampled between 1 and 50 MB and with distinct random seeds to ensure content diversity.
- *x264* **(video processing)** is a video encoder that takes a raw YUV video file as input, along with parameters specifying frame resolution and frame count. To construct inputs representative of real-time workloads, we used videos from the YouTube User Generated Content (UGC) dataset [59], which captures realistic distortions and motion patterns. We selected videos from the 360p subset and split each video into chunks of at most five frames, yielding short encoding tasks suitable for real-time execution.

- *exchange2* **(recursive solution generator)** is a Fortran program that takes as input a Sudoku puzzle and generates new puzzles through structured transformations. Each input is a partially filled 9×9 Sudoku grid. We synthesized 200 input puzzles with guaranteed unique solutions, with the number of empty cells sampled uniformly from $[50, 57]$ to control puzzle difficulty. Puzzles are encoded as a flattened 81-character string.
- *xz* **(compression)** performs lossless data compression and decompression. Each input consists of a file to be compressed, along with metadata such as file size, SHA-512 hash, expected compressed-size bounds, and a compression level between 0 and 9. We drew inputs from standard compression benchmarks, including the Silesia [17], Canterbury, Calgary, Artificial, Miscellaneous, and Large corpora [3], covering both typical and worst-case compression behaviors. To enforce a global size constraint, files larger than 1 MB were split into 1 MB chunks, and the compression level was fixed to 4.

7.2 Data Collection and Data Augmentation

Platform. We conducted all experiments on a CAT-enabled platform equipped with an Intel Xeon E5-2618L v3 processor featuring 8 cores, 20 MB 20-way set-associative L3 cache, and single-channel 8 GB DDR4-2133 DRAM. We considered $B = 2$ shared resources: the shared L3 cache and memory bandwidth. We used Cache Allocation Technology (CAT) [32] to partition the cache and MemGuard [63] to regulate memory bandwidth. CAT divides the L3 cache into 20 partitions, with a minimum allocation of 2 partitions per core on our platform. We similarly divided the platform's guaranteed memory bandwidth of 1.4 GB/s into 20 partitions of 70 MB/s each. We disabled hardware cache prefetching and hyperthreading.

Empirical profiling and data collection. To obtain execution profiles for each benchmark, we pinned it to a dedicated core using `chrt`. For every input in its generated input set, we executed the benchmark 10 times under each static allocation β of cache and memory bandwidth. During each execution run, we sampled CPU performance counters at a fixed interval of $\Delta = 10$ ms and recorded the number of retired instructions in each interval. To feasibly profile all benchmarks, we enforced equal allocations across resource types, restricting β to the set $\mathcal{B} = \{(2, 2), (3, 3), \ldots, (20, 20)\}$. With 200 inputs per benchmark, this procedure yields $10 \cdot 200 \cdot 19 = 38{,}000$ static execution profiles per benchmark.

Data augmentation. Static profiles assume fixed resource allocations, whereas at runtime resource allocations may change dynamically. To obtain sufficient training data without explicitly profiling all possible allocation sequences (which is impractical), we synthesize dynamic profiles for a policy that reallocates resources at every sampling point as follows.

Let x_k^P denote the cumulative retired-instruction count at the k-th sampling (and reallocation) point, at time $t = k\Delta$, for a static profile P. Starting from an initial static profile P_0 – collected for some input I under allocation β_0 – we randomly select a new allocation $\beta_1 \in \mathcal{B}$ for the first reallocation point $k = 1$. Among static profiles for the same input under allocation β_1, we select the profile P_1 whose $x_k^{P_1}$ is closest to $x_k^{P_0}$. We then switch to P_1, randomly select a new allocation $\beta_2 \in \mathcal{B}$ for the next reallocation point $k = 2$. We continue performing this random walk over static profiles until no further sampling point exists (i.e., the program terminates). This procedure produces one synthesized dynamic profile per static profile, yielding an additional 38,000 dynamic profiles per benchmark.

Our approach leverages two properties to generate *valid* profiles under dynamic allocation. First, as our programs deterministically follow the same execution path for a given input, profiles from separate runs with identical input can be aligned via cumulative instruction

Table 1 Selected lag w_i and input features f_i for each benchmark.

Benchmark	w_i	f_i	Input Features
mcf	4	2	Number of nodes, number of edges
omnetpp	5	1	Total number of hosts
xalancbmk	4	1	File size
x264	5	4	Number of frames, frame area, spatial complexity, temporal motion
exchange2	3	2	Number of blanks, average constraint tightness
xz	4	1	File size

counts. Second, prior work [26, 20] shows that reallocating LLC partitions via CAT and memory bandwidth via MemGuard incurs negligible overhead relative to the 10 ms sampling interval. Thus, profiles can be composed across allocations without modeling reconfiguration costs. For smaller windows, or to account for transient cache reconfiguration effects, overheads can be incorporated by scaling retired-instruction counts based on working-set size, memory behavior, and the magnitude of allocation changes [51].

7.3 Feature Selection

For practical implementation, we define the state vector $x_{i,j,k} \in \mathbb{R}^{p_i}$ of each job $\tau_{i,j}$ at allocation step $k \in \mathbb{N}_0$, i.e., at time $t_k := t_0 + k\Delta$, as (i) the total number of retired instructions by time t_k, (ii) $f_i \in \mathbb{N}^+$ features descriptive of the input to $\tau_{i,j}$, where f_i is task-dependent, and (iii) a lag of resource allocations and cumulative number of retired instructions for $\tau_{i,j}$ in the previous $w_i \in \mathbb{N}_0$ allocation steps, where w_i is also task-dependent. Together, these $p_i = 1 + f_i + w_i(B + 1)$ features capture the current execution state of a job.

Table 1 summarizes the numbers of lag windows w_i and input features f_i used for each SPEC benchmark τ_i. For each benchmark, w_i was selected via grid search over $1 \leq w_i \leq 5$ to maximize the prediction accuracy of μ_i and e_i. To limit runtime overhead, input features were chosen based on benchmark semantics and were restricted to quantities that can be extracted efficiently at runtime, without requiring expensive static or dynamic analysis.[4]

For *x264*, spatial complexity and temporal motion are defined as the standard deviation and the mean absolute difference of pixel values across frames, respectively. These features can be computed with negligible overhead, as file I/O dominates the job startup time. For *exchange2*, we define the initial constraint tightness of each empty cell of the puzzle as the number of distinct non-empty cells in its row, column and box union. The feature value is then computed as the average constraint tightness across all empty cells.

7.4 Model Evaluation

We leveraged XGBoost (eXtreme Gradient Boosting) [12], a learning framework that builds an additive ensemble of regression trees via gradient boosting. We trained a separate XGBoost regressor for each prediction target using the squared-error objective. Trees were kept deliberately shallow (maximum depth = 3) and the ensemble was limited to at most 64 boosting rounds, with early stopping after 20 rounds of no improvement on a validation set. We used a relatively high learning rate (eta = 0.30) to compensate for the reduced capacity. These choices deliberately prioritize a compact, low-latency model over maximal accuracy.

[4] We refer the reader to [33] for a principled approach to determine these features offline.

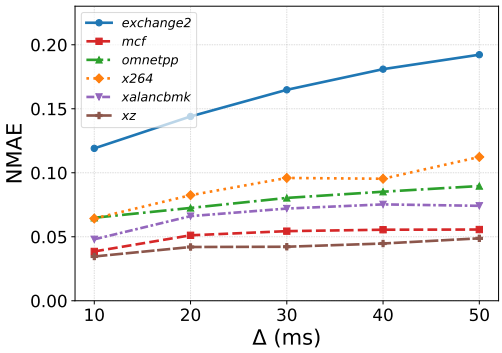


Figure 2 NMAE (μ_i) vs. Δ per benchmark.

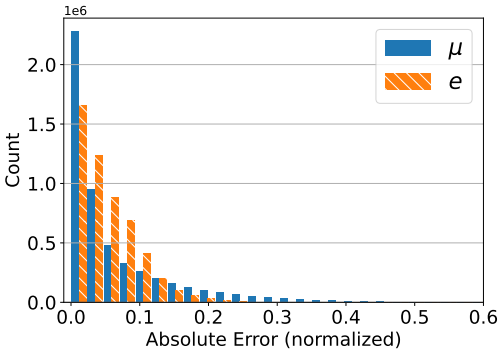


Figure 3 Error distribution for *omnetpp*.

Table 2 Performance of GBRT models on SPEC benchmarks for $\Delta = 10$ ms. R^2 and the normalized mean absolute error (MAE) were computed across all test inputs and resource allocations.

Task	R^2		NMAE		Model Size (KB)		Pred. Time (μs)	
	e_i	μ_i	e_i	μ_i	e_i	μ_i	e_i	μ_i
mcf	0.7485	0.7838	0.0545	0.0385	6.31	7.69	1.18	1.20
omnetpp	0.8216	0.7135	0.0514	0.0648	5.60	7.82	1.05	1.19
xalancbmk	0.9746	0.9038	0.0212	0.0480	5.95	7.76	1.16	1.23
x264	0.1688	−0.5331	0.0903	0.0643	4.39	3.06	0.86	0.75
exchange2	−0.0277	−0.1160	0.1088	0.1191	3.80	2.99	0.79	0.73
xz	0.4769	0.8783	0.0708	0.0346	6.08	7.76	1.09	1.11

To evaluate generalization across unseen program inputs, we employed an input-based train/test split. Following [33], we used only 10% of our input set for training. Specifically, we clustered the cumulative instruction count of each input and selected the median of each cluster to include in the training set. We used an additional 10% for validation, resulting in a held-out test set consisting of 80% of our generated inputs. Across all six benchmarks, the resulting datasets contain approximately 2.3-7.6 million data points in total.

For each benchmark, we trained 5 different models: one for each value of Δ, where $\Delta \in \{10, 20, 30, 40, 50\}$ milliseconds. Fig. 2 shows the model accuracy, reported using the Normalized Mean Absolute Error (NMAE), vs. Δ, where the NMAE was computed on the set of held-out test inputs for each benchmark. Notice that the prediction error in μ_i generally increases as Δ increases.[5] Therefore, we set $\Delta = 10$ to report additional results, including the coefficient of determination (R^2), the model size, and the median per-prediction latency of our fixed-point C implementation (for in-kernel deployment). We observed negligible accuracy loss (at most 0.003% across all benchmarks) when moving to fixed-point inference.

Table 2 summarizes accuracy, size, and speed results, while Fig. 3 provides a more detailed picture of the error distribution for *omnetpp*. The models perform best on benchmarks exhibiting distinct execution phases and/or strong resource dependencies, including *xalancbmk*, *omnetpp* and *mcf*. In contrast, *exchange2* and *x264* exhibit weak R^2 results. These benchmarks are highly CPU-intensive and show limited phase structure or discernible resource dependency in their execution profiles, leaving less execution state signal for the model to exploit. The resulting model sizes range from 2.99 to 7.82 KB while the per-sample inference latency in the C implementation ranges from 0.73 to 1.23 μs, allowing us to achieve low-overhead fine-grained allocation while leveraging predictions.

[5] We observed that the prediction error in e_i remains approximately the same across all evaluated Δ, for each benchmark τ_i.

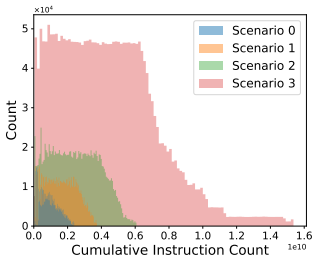


Figure 4 Distribution of $x^{(1)}$ for each scenario (*x264*).

Table 3 Empirical coverage for weighted CP across different operating scenarios (S0 - S3).

Task	S0		S1		S2		S3	
	e_i	μ_i	e_i	μ_i	e_i	μ_i	e_i	μ_i
mcf	0.93	0.89	0.94	0.85	0.99	0.88	0.84	0.92
omnetpp	0.81	0.88	0.94	0.91	0.80	0.90	0.68	0.89
xalancbmk	0.87	0.88	0.82	0.86	0.95	0.92	0.70	0.88
x264	1.00	0.89	1.00	0.89	1.00	0.98	0.95	0.99
exchange2	0.98	0.83	0.79	0.92	0.84	0.91	0.90	0.90
xz	0.68	0.92	0.79	0.86	0.99	0.90	0.93	0.85

7.5 Weighted CP Evaluation

To evaluate the effectiveness of our weighted conformal prediction, for each SPEC benchmark τ_i, we clustered its test inputs into groups based on their execution behavior, where each cluster represents a distinct operating scenario of the system. Specifically, we used the total number of retired instructions under each input as the clustering feature. For each scenario, we evenly partition the data into three disjoint sets: $\mathcal{D}_i^{\text{wt}}$, $\mathcal{D}_i^{\text{cal}}$, and $\mathcal{D}_i^{\text{test}}$. To solve (10) – that is, to learn the weights that maximize the empirical coverage guarantee for $\mathcal{D}_i^{\text{cal}}$ given q_i^* (the empirical $(1-\delta_i)^{\text{th}}$-quantile of the nonconformity scores in $\mathcal{D}_i^{\text{wt}}$) – we trained a small neural network. For each τ_i, we set the miscoverage rate $\delta_i = 0.1$ and clustered the test input set (consisting of 180 inputs) into four operating scenarios.

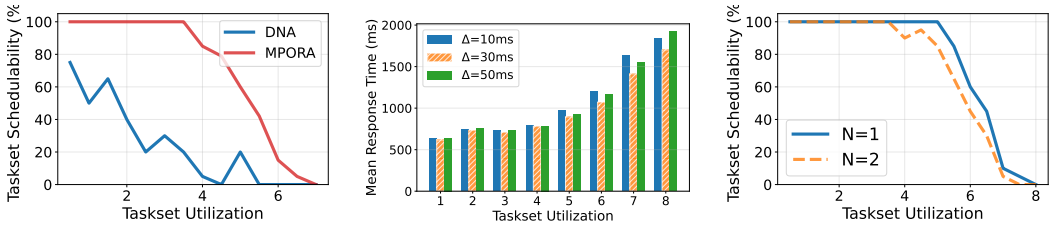
To give a concrete example, each scenario for benchmark *x264* corresponds to a distinct frame-processing rate. Figure 4 shows how the distribution of $x^{(1)}$ depends on this operational state. The goal of weighted conformal prediction is to adapt to such distribution shift. Table 3 reports the empirical coverage of the weighted CP bounds across all six benchmarks, both prediction targets, and four operating scenarios, with a miscoverage rate of $\delta_i = 0.1$. Notice that across all tasks and scenarios, the empirical coverage of $\widehat{e}_i$ and $\widehat{\mu}_i$ remains generally close to the nominal target $1 - \delta_i = 0.9$, which experimentally validates that the learned importance weights successfully concentrates mass on calibration samples representative of the runtime distribution of each scenario, thereby minimizing the $\mathrm{d_{TV}}$ penalty term in (7).

For some tasks and scenarios, e.g. *omnetpp* and *xalancbmk* in Scenario 3, we observe a coverage below the nominal target for e_i. Since Scenario 3 corresponds to inputs that induce the largest execution times, we conjecture that long tails (e.g., see Figure 4) in the state distributions lead to a large $\mathrm{d_{TV}}$ distance for which the weights cannot fully compensate. Introducing another scenario in these cases, collecting a more representative calibration set for the tail region, or increasing the miscoverage rate for these benchmarks, would be sufficient to recover nominal coverage. Finally, notice that coverage remains generally consistent across scenarios without retraining the predictive models, indicating that the scenario-specific weights are sufficient to adapt the uncertainty quantification to distribution shift.

8 Prototype and Experimental Evaluation

8.1 Linux Kernel Integration and Experiment Setup

Prototype. To evaluate our method end-to-end, we implemented a prototype of MPORA in Linux 6.12 with the real-time patch `PREEMPT_RT` enabled. At each allocation step k, the kernel module measures the current execution state of each scheduled job $\tau_{i,j}$, predicts the number of retired instructions and remaining execution time using the learned models $\widehat{\mu}_i$



(a) MPORA-DNA schedulability. **(b)** Effect of Δ on MPORA. **(c)** Effect of N on MPORA.

Figure 5 MPORA prototype evaluation.

and $\widehat{e}_i$ under each possible allocation, solves (3) to determine the optimal allocation, then applies the allocation for that step. MPORA runs on a single core to minimize interference. For small N, we solve (3) by enumerating all feasible resource assignments; if no feasible solution exists, we select the allocation that minimizes total job tardiness.

In addition to the kernel module, we added approximately 150 lines of Linux in-tree code to track each task's metadata, including the current instruction count, the lag of instructions retired and resource allocations, and per-job input features (passed from user space via `prctl` once per job). We used Intel CAT [32] and MemGuard [63] for cache and memory bandwidth assignments, respectively. We additionally modified MemGuard's implementation to support `SCHED_DEADLINE` policy in Linux, enabling tasksets to run under global EDF scheduling.

Experiment setup. We evaluated our solution on our prototype, using the same platform setup as in Sec. 7.2. We used $M = 4$ cores and the allocation set $\mathcal{B} = \{(2,2),(3,3),\ldots,(20,20)\}$. Thus, for a horizon length of $N = 1$, computing the optimal allocation at each step requires $(M|\mathcal{B}|)^N = 4 \times 19 = 76$ predictions per target (i.e., 152 predictions total).

Our workloads consist of randomly generated tasksets executing programs from the SPEC benchmark suite. We generated 20 tasksets at each utilization step of 0.5 up to 8.0. Given a target taskset utilization, we uniformly sampled per-task utilizations from [0.1, 0.5], rounding down the final task's utilization if needed to meet the target. Each task was assigned a random benchmark from the set of SPEC benchmarks evaluated in Sec. 7. A task's *reference* execution time is defined as the median execution time observed across all its profiling runs and all inputs. Its period (deadline) was set to its reference WCET divided by its utilization. At runtime, job inputs were selected from the input sets described in Sec. 7.1.

Baseline. We compared MPORA against DNA [26], a dynamic resource allocation technique for real-time systems. DNA computes lookup tables offline by clustering profile data into phases, demarcates these phases by cumulative instruction count, and assigns an average instruction rate to each phase. It then leverages heuristics to iteratively allocate resources online. Since DNA assumes a fixed execution path, we constructed its phase-based lookup tables using the worst-case execution path induced by our set of possible inputs.

8.2 Prototype Evaluation Results

Schedulability vs. baseline. Fig. 5a shows the percent of tasksets that were empirically schedulable for MPORA ($\Delta = 10$ ms, $N = 1$) vs. DNA ($\Delta = 10$ ms). MPORA schedules much larger workloads than DNA, due to its ability to adapt resource allocations to input-dependent execution. Across all utilizations, MPORA scheduled 3× more tasksets.

Runtime overhead. To evaluate MPORA's runtime overhead with these parameters ($\Delta = 10$, $N = 1$), we measured end-to-end resource allocation costs (including predictions, allocation computation, and resource reconfiguration). Across 20,000 observations, the minimum, maximum, and mean costs per allocation were 22 μs, 367 μs and 117 μs, respectively. Thus, invoking MPORA every $\Delta = 10$ ms on one of four cores corresponds to maximum and average overheads of 0.92% and 0.29%, respectively.

Effect of Δ. Fig. 5b shows the mean response time at each utilization step when running MPORA with $\Delta = \{10, 30, 50\}$ ms and $N = 1$. Notice that the response time for $\Delta = 10$ grows the fastest as load increases, demonstrating the negative effect of frequent resource re-allocation at heavy loads. However, $\Delta = 30$ performs better than $\Delta = 50$ across all utilizations, reaffirming the importance of fine-grained, reactive resource allocation.

Effect of N. Fig. 5c shows taskset schedulability vs. utilization for both $N = 1$ and $N = 2$ (keeping $\Delta = 30$ ms fixed). Even if we prune the search space, we found that increasing the prediction horizon results in worse schedulability, as the number of predictions grows exponentially in N. Solving (3) efficiently for larger N is an important future direction.

9 Conclusion

We presented the design, kernel-level implementation, and evaluation of MPORA, an uncertainty-aware dynamic resource allocation framework for multi-path, input-dependent real-time tasks on multicore platforms. By integrating learned predictive models with a receding-horizon MPC-like controller and weighted conformal prediction, MPORA adapts resource allocations online while enforcing deadline constraints robust to unseen execution paths and distribution shifts. Evaluations show MPORA achieves high prediction accuracy with low runtime overhead, while improving schedulability and response times over existing techniques. More broadly, MPORA demonstrates that uncertainty-aware learning and resource control can operate within the OS kernel, enabling self-adaptive real-time systems.

References

1 Liviu Aolaritei, Zheyu Oliver Wang, Julie Zhu, Michael I. Jordan, and Youssef M. Marzouk. Conformal Prediction under Lévy-Prokhorov Distribution Shifts: Robustness to Local and Global Perturbations. *arXiv preprint*, 2025. doi:10.48550/arXiv.2502.14105.

2 ARM. PrimeCell Level 2 Cache Controller (PL310) – Technical Reference Manual. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0246c/index.html.

3 Ross Arnold and Tim Bell. The Canterbury Corpus. https://corpus.canterbury.ac.nz/, 1997.

4 Jonathan Bader, Fabian Lehmann, Lauritz Thamsen, Ulf Leser, and Odej Kao. Lotaru: Locally Predicting Workflow Task Runtimes for Resource Management on Heterogeneous Infrastructures. *Future Generation Computer Systems*, 150:171–185, 2024. doi:10.1016/J.FUTURE.2023.08.022.

5 Jonathan Bader, Kathleen West, Soeren Becker, Svetlana Kulagina, Fabian Lehmann, Lauritz Thamsen, Henning Meyerhenke, and Odej Kao. Predicting the Performance of Scientific Workflow Tasks for Cluster Resource Management: An Overview of the State of the Art, 2025. doi:10.48550/arXiv.2504.20867.

6 Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Conformal Prediction beyond Exchangeability. *Annals of Statistics*, 51(2):816–845, 2023.

7 Arnamoy Bhattacharyya, Stelios Sotiriadis, and Cristiana Amza. Online Phase Detection and Characterization of Cloud Applications. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, pages 98–105. IEEE, 2017. doi:10.1109/CLOUDCOM.2017.21.

8 G. A. Bondar, A. Eisenklam, Y. Cai, R. Gifford, T. Sial, L. T. X. Phan, and A. Halder. Generative Profiling for Soft Real-Time Systems and Its Applications to Resource Allocation. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, 2026.

9 Maxime Cauchois, Suyash Gupta, Alnur Ali, and John C. Duchi. Robust Validation: Confident Predictions Even When Distributions Shift. *Journal of the American Statistical Association*, 119(548):3033–3044, 2024.

10 Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery D. Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s):94:1–94:26, 2013. doi:10.1145/2465787.2465796.

11 Kong Yao Chee, M. Ani Hsieh, and George J. Pappas. Uncertainty Quantification for Learning-Based MPC Using Weighted Conformal Prediction. In *Proceedings of the IEEE Conference on Decision and Control*, pages 342–349, 2023. doi:10.1109/CDC49753.2023.10383587.

12 Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016. doi:10.1145/2939672.2939785.

13 Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-Path Programs. In Robert Davis, editor, *Proceedings of the Euromicro Conference on Real-Time Systems*, 2012.

14 Laurent David and Isabelle Puaut. Static Determination of Probabilistic Execution Times. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 223–230, 2004. doi:10.1109/ECRTS.2004.34.

15 Robert I. Davis and Alan Burns. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Comput. Surv.*, 43(4), 2011. doi:10.1145/1978802.1978814.

16 Robert I. Davis and Liliana Cucu-Grosjean. A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 6(1):03:1–03:60, 2019. doi:10.4230/LITES-V006-I001-A003.

17 Sebastian Deorowicz. Silesia Compression Corpus. https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia, 2003.

18 Stewart Edgar and Alan Burns. Statistical Analysis of WCET for Scheduling. In *Proceedings of the Real-Time Systems Symposium*, pages 215–224, 2001. doi:10.1109/REAL.2001.990614.

19 Abigail Eisenklam. MPORA. Software, swhId: swh:1:dir:effbb0b80f77dea1364355a09ce8c305aebc15b2 (visited on 2026-06-22). URL: https://github.com/phan-lab/MPORA, doi:10.4230/artifacts.26769.

20 Abigail Eisenklam, Robert Gifford, Georgiy A Bondar, Yifan Cai, Tushar Sial, Linh Thi Xuan Phan, and Abhishek Halder. Rasco: Resource Allocation and Scheduling Co-Design for DAG Applications on Multicore. *ACM Trans. Embed. Comput. Syst.*, 24(5s), 2025. doi:10.1145/3761814.

21 Shereen Elsayed, Daniela Thyssens, Ahmed Rashed, Hadi Samer Jomaa, and Lars Schmidt-Thieme. Do We Really Need Deep Learning Models for Time Series Forecasting?, 2021. arXiv:2101.02118.

22 Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

23 Xing Fu, Khairul Kabir, and Xiaorui Wang. Cache-Aware Utilization Control for Energy Efficiency in Multi-Core Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 102–111, 2011. doi:10.1109/ECRTS.2011.18.

24 Asaf Gendler, Tsui-Wei Weng, Luca Daniel, and Yaniv Romano. Adversarially Robust Conformal Prediction. In *International Conference on Learning Representations*, 2022.

25 Robert Gifford, Felipe Galarza-Jimenez, Linh Thi Xuan_Phan, and Majid Zamani. Decntr: Optimizing Safety and Schedulability with Multi-Mode Control and Resource Allocation Co-Design. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 306–319, 2024. doi:10.1109/RTAS61025.2024.00032.

26 Robert Gifford, Neeraj Gandhi, Linh Thi Xuan Phan, and Andreas Haeberlen. DNA: Dynamic Resource Allocation for Soft Real-Time Multicore Systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 196–209, 2021. doi:10.1109/RTAS52030.2021.00024.

27 Robert Gifford and Linh Thi Xuan Phan. Multi-Mode on Multi-Core: Making the Best of Both Worlds with Omni. In *Proceedings of the Real-Time Systems Symposium*, pages 118–131, 2022. doi:10.1109/RTSS55097.2022.00020.

28 Jingzhi Gong and Tao Chen. Predicting Software Performance with Divide-and-Learn. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 858–870, 2023. doi:10.1145/3611643.3616334.

29 Sandro Grebant, Clément Ballabriga, Julien Forget, and Giuseppe Lipari. WCET Analysis with Procedure Arguments as Parameters. In *Proceedings of the International Conference on Real-Time Networks and Systems*, pages 11–22, 2023. doi:10.1145/3575757.3593655.

30 Huong Ha and Hongyu Zhang. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. In *Proceedings of the International Conference on Software Engineering*, pages 1095–1106, 2019. doi:10.1109/ICSE.2019.00113.

31 Damien Hardy and Isabelle Puaut. Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches. *Proceedings of the International Conference on Real-Time Networks and Systems*, 51(2):128–152, 2015. doi:10.1007/S11241-014-9212-X.

32 Intel. Improving Real-Time Performance by Utilizing Cache Allocation Technology, April 2015. White Paper.

33 Yongin Kwon, Sangmin Lee, Hayoon Yi, Donghyun Kwon, Seungjun Yang, Byung-gon Chun, Ling Huang, Petros Maniatis, Mayur Naik, and Yunheung Paek. Mantis: Efficient Predictions of Execution Time, Energy Usage, Memory Usage and Network Usage on Smart Mobile Devices. *IEEE Transactions on Mobile Computing*, 14(10):2059–2072, 2015. doi:10.1109/TMC.2014.2374153.

34 Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018.

35 Alberto Leva, Alessandro Vittorio Papadopoulos, and Martina Maggio. A General Control-Theoretical Methodology for Runtime Resource Allocation in Computing Systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3487–3492, 2013. doi:10.1109/CDC.2013.6760418.

36 Yun Liang and Tulika Mitra. Cache Modeling in Probabilistic Execution Time Analysis. In Limor Fix, editor, *Proceedings of the ACM/IEEE Design Automation Conference*, pages 319–324, 2008. doi:10.1145/1391469.1391551.

37 George Lima and Iain Bate. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In Gabriel Parmer, editor, *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 187–198, 2017. doi:10.1109/RTAS.2017.17.

38 George Lima, Dario Dias, and Edna Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 200–211, 2016. doi:10.1109/ECRTS.2016.20.

39 Lars Lindemann, Yiqi Zhao, Xinyi Yu, George J. Pappas, and Jyotirmoy V. Deshmukh. Formal Verification and Control with Conformal Prediction: Practical Safety Guarantees for Autonomous Systems. *IEEE Control Systems*, 45:72–122, 2025.

40 Erika Susana Alcorta Lozano and Andreas Gerstlauer. Learning-Based Phase-Aware Multi-Core CPU Workload Forecasting. *ACM Trans. Des. Autom. Electron. Syst.*, 28(2):1–27, 2022. doi:10.1145/3564929.

41 Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems*, 23(1):85–126, 2002. doi:10.1023/A:1015398403337.

42 Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-Time Cache Management Framework for Multi-Core Architectures. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 45–54, 2013. doi:10.1109/RTAS.2013.6531078.

43 Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive Confidence Machines for Regression. In *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages 345–356, 2002. doi:10.1007/3-540-36755-1_29.

44 Jinsu Park, Seongbeom Park, and Woongki Baek. CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers. In *EuroSys*, 2019.

45 Tirthak Patel and Devesh Tiwari. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 193–206, 2020. doi:10.1109/HPCA47549.2020.00025.

46 Raghavendra Pradyumna Pothukuchi, Amin Ansari, Petros Voulgaris, and Josep Torrellas. Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture*, pages 658–670, 2016. doi:10.1109/ISCA.2016.63.

47 R. Tyrrell Rockafellar and Roger J-B Wets. *Variational Analysis*. Springer, Berlin, Heidelberg, 1998. doi:10.1007/978-3-642-02431-3.

48 Ahsan Saeed, Dakshina Dasari, Dirk Ziegenbein, Varun Rajasekaran, Falk Rehm, Michael Pressler, Arne Hamann, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. Memory Utilization-Based Dynamic Bandwidth Regulation for Temporal Isolation in Multi-Cores. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 133–145, 2022. doi:10.1109/RTAS54340.2022.00019.

49 Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases*, pages 974–985, 2002. doi:10.1016/B978-155860869-6/50096-2.

50 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. E-WarP: A System-Wide Framework for Memory Bandwidth Profiling and Management. In *Proceedings of the Real-Time Systems Symposium*, pages 345–357, 2020. doi:10.1109/RTSS49844.2020.00039.

51 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. Profile-Driven Memory Bandwidth Management for Accelerators and CPUs in QoS-Enabled Platforms. In *Real-Time Systems*, volume 58, pages 235–274, 2022. doi:10.1007/S11241-022-09382-X.

52 Standard Performance Evaluation Corporation. SPEC CPU® 2017 Benchmark. https://www.spec.org/cpu2017/.

53 J.A. Stankovic, Chenyang Lu, S.H. Son, and Gang Tao. The Case for Feedback Control Real-Time Scheduling. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 11–20, 1999.

54 Srinivasan Subramaniyan and Xiaorui Wang. FC-GPU: Feedback Control GPU Scheduling for Real-Time Embedded Systems. *ACM Trans. Embed. Comput. Syst.*, 24(5s), 2025. doi:10.1145/3761812.

55 Binqi Sun, Zhihang Wei, Andrea Bastoni, Debayan Roy, Mirco Theile, Tomasz Kloda, Rodolfo Pellizzoni, and Marco Caccamo. Multi-Objective Memory Bandwidth Regulation and Cache Partitioning for Multicore Real-Time Systems. In Renato Mancuso, editor, *Proceedings of the Euromicro Conference on Real-Time Systems*, 2025.

56 Karl Taht, James Greensky, and Rajeev Balasubramonian. The POP Detector: A Lightweight Online Program Phase Detection Framework. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 48–57, 2019. doi:10.1109/ISPASS.2019.00013.

57 Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018.

58 Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer International Publishing, 2nd edition, 2022.

59 Yilin Wang, Sasi Inguva, and Balu Adsumilli. YouTube UGC Dataset for Video Compression Research. In *Proceedings of the International Workshop on Multimedia Signal Processing*, pages 1–5, 2019. doi:10.1109/MMSP.2019.8901772.

60 Meng Xu, Robert Gifford, and Linh Thi Xuan Phan. Holistic Multi-Resource Allocation for Multicore Real-Time Virtualization. In *Proceedings of the Design Automation Conference*, pages 1–6, 2019.

61 Meng Xu, Linh Thi Xuan Phan, H. Choi, Y. Lin, H. Li, C. Lu, and Insup Lee. Holistic Resource Allocation for Multicore Real-Time Systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 345–356, 2019.

62 Rui Xu, Chao Chen, Yue Sun, Parvathinathan Venkitasubramaniam, and Sihong Xie. Wasserstein-Regularized Conformal Prediction under General Distribution Shift. In *The Thirteenth International Conference on Learning Representations*, 2025.

63 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory Bandwidth Management for Efficient Performance Isolation in Multi-Core Platforms. *IEEE Transactions on Computers*, 65(2):562–576, 2016. doi:10.1109/TC.2015.2425889.