Switch Precision Where It Pays: An Optimal Scheduling Approach for Efficient Control

Debarpita Banerjee, Debasmita Lohar, and Sumana Ghosh

***Abstract*—Modern cyber-physical systems, such as automotive control, rely on feedback controllers, running in closed loops, that regulate the system's output towards a desired reference. In practice, however, the controller must also be scheduled efficiently on resource-constrained processors, where the choice of numerical precision for controller implementation directly affects both control quality and computational cost. This trade-off is critical: higher precision improves control performance but increases runtime, while lower precision executes faster in the processor but may degrade overall system performance.**

We present the first closed-loop control scheduling framework with per-sample precision switching that automatically decides *when* to execute the controller in *which* floating-point precision to balance control performance and computational efficiency while ensuring that the system output remains within a specified reference band. We cast this problem as a multi-objective optimization and formulate it as a Mixed-Integer Quadratic Program (MIQP) with sound linearizations and worst-case roundoff error bounds that capture precision-dependent effects. The resulting MIQP is then solved using state-of-the-art solvers. Experiments on standard benchmark control systems show that switching between 32-bit and 16-bit floating-point implementations yields an average runtime reduction of 5.8% compared to always using 32-bit execution, and a 24.8% improvement in control performance compared to always using 16-bit execution, while maintaining near-optimal overall performance. We also show that optimized switching offers a more principled trade-off between control performance and runtime than simple heuristic precision-switching strategies. Finally, we demonstrate that precision-switching can improve the schedulability of multiple control tasks on a processor, reflecting its usage in real-world applications.

***Index Terms*—Plant-control systems, real-time scheduling, floating-point arithmetic, control performance**

I. Introduction

Modern cyber-physical systems, such as those in automotive control, avionics, and robotics, widely employ feedback controllers that run in closed loops: they continuously adjust control inputs based on the systems' states to steer systems towards their desired references. These controllers are often executed as real-time control tasks on the processor, which must run periodically and complete within strict timing requirements or deadlines to preserve control performance and responsiveness. Therefore, designing effective scheduling strategies for controllers remains an important problem.

There exists a large body of work on real-time scheduling of controllers [35], [28], [22] and co-scheduling of multiple controllers [34], [40], [20], [11], exploring various aspects like control performance, stability under overload scenarios [20], [7], and platform-level uncertainties [38], [22]. However, this literature usually overlooks practical controller implementations that can influence scheduling and control performance.

In practice, controllers are implemented in finite-precision arithmetic (e.g., floating-point or fixed-point), and the chosen numerical precision directly affects both computational cost and control performance. Lower precision reduces runtime and may therefore improve schedulability, but accumulated round-off errors degrade numerical accuracy and may compromise control performance. In contrast, higher precision improves numerical accuracy and often improves control performance, but increases runtime and can tighten scheduling margins. Balancing this trade-off is thus essential for achieving both correct and efficient control.

Recently, several works have considered the role of numerical precision in control-theoretic contexts such as robust controller synthesis [32], safety analysis [37], [36], and mixed-precision quantization of neural network controllers [27]. However, none of them addresses precision as a scheduling decision or explores how it affects performance requirements when controllers are executed as real-time tasks on a processor.

To bridge this gap, we introduce, to the best of our knowledge, the first closed-loop control scheduling framework with per-sample precision switching. The key idea is: for a given controller, rather than fixing a single floating-point precision a priori for all samples (i.e., discrete time steps for control execution), our framework computes a schedule that selects one (uniform) floating-point precision implementation per sample from the available options, with precision changes permitted only between consecutive samples. Thus, the framework synthesizes *precision-switching sequences* that alternate precisions across samples to reduce computational cost while preserving the desired control performance.

We emphasize that the novelty of this work is not the general idea of using multiple precisions, which has been widely studied in several other computational settings, including mixed-precision iterative refinement for numerical linear algebra [9], [10]. Rather, our contribution is to formulate precision selection as part of the closed-loop control scheduling decision. In contrast to mixed-precision techniques that combine multiple precisions within a program or algorithm, our framework schedules *only* one precision per control sample and optimizes switching across samples while accounting for control performance, execution time, and switching overheads.

We formulate this scheduling problem as a multi-objective *mixed-integer quadratic program* (MIQP) that jointly op-

Debarpita Banerjee and Sumana Ghosh are with the Computer and Communication Sciences Division, Indian Statistical Institute Kolkata, India (e-mail: debarpita2023_r@isical.ac.in, sumana@isical.ac.in).

Debasmita Lohar is with the Software Engineering Section, IT University of Copenhagen, Denmark (e-mail: debloh@itu.dk).

timizes: (i) *control performance*, quantified by the linear quadratic cost LQR [30] (minimizing the *control effort*), and (ii) *execution time of the resulting schedule*, quantified by per-sample runtimes and switching overheads incurred due to precision changes. The optimization is subject to the *settling-time requirement*: the system output must reach its reference within the stipulated settling time, thereby capturing the system's responsiveness. To faithfully model finite-precision execution, we incorporate worst-case floating-point roundoff errors on inputs as well as on the computed states and outputs.

A direct encoding of these requirements introduces nonlinear constructs that are computationally intractable. We address this by applying exact linearizations where possible and model roundoff errors using sound worst-case bounds. This yields an MIQP with a convex objective and entirely linear constraints, which can be solved efficiently by state-of-the-art MIQP solvers. By formulating precision switching as an optimization problem, our approach avoids a computationally expensive exhaustive search over precision choices at each sample. While this paper focuses on floating-point controller implementations, the proposed technique is generic and can be applied to other finite-precision formats.

We implemented our approach in a prototype tool called *Apex* that takes controller design parameters, timing metrics, and the available precision options as inputs and returns an optimized precision-switching sequence. We evaluate it on standard automotive control benchmarks, synthesizing sequences that switch between 16-bit and 32-bit floating-point implementations.

Our evaluation shows that precision switching can deliver the best of both worlds by using lower precision when it is safe and reserving higher precision only when needed. On average, executing the controller according to the synthesized switching sequences reduces total runtime by 5.8% compared to executing entirely in 32-bit, while improving control performance by 24.8% compared to executing entirely in 16-bit, achieving near–32-bit performance overall. We also find that switching overhead accounts for only about 0.01% of the total execution time, making precision switching a practical deployment option. The comparison with simple heuristic switching strategies further shows that optimized switching provides a more systematic way to navigate the control performance-runtime trade-off than fixed or random switching. Finally, we demonstrate that precision switching can benefit multi-task settings: by reducing execution time (and thus processor utilization) on a uniprocessor, it can schedule tasks that are otherwise infeasible with only one high-precision option.

***Contributions**:* To summarize, this paper makes the following contributions.

1) We introduce the first control scheduling framework with per-sample precision switching, that synthesizes precision-switching sequences for controller implementations to balance control performance and execution time.
2) We formulate the scheduling as a multi-objective MIQP that jointly optimizes LQR cost and execution time while enforcing a settling-time requirement, and incorporating worst-case roundoffs for floating-point implementations.
3) We have implemented our approach in a prototype tool, *Apex*, which is publicly available on GitHub: https://github.com/DBanerjee19/Apex.
4) We evaluate Apex on standard automotive control benchmarks and demonstrate its effectiveness w.r.t. entirely uniform-precision baselines and three simple heuristic strategies based on fixed or random switching patterns. We also highlight its implications for multi-task scheduling on shared processors.

II. Background

This section provides an overview of real-time control systems and the fundamentals of floating-point arithmetic, which will be used throughout the paper.

A. Real-Time Closed-Loop Control Systems

A plant-control system is a closed-loop continuous system in which a dynamical process (*plant*) is regulated by a feedback controller. The controller observes the plant's output and computes a control input to maintain a desired reference. In embedded processors, a controller is implemented as a software control task that executes periodically with a fixed *sampling period*. The dynamics are thus represented in discrete time (considering linear time-invariant (LTI) models) as:

$$x[k+1] = Ax[k] + Bu[k], \quad y[k] = Cx[k], \tag{1}$$

where $x[k], u[k], y[k]$ are the discrete-time state, control input, and output at the k-th sampling instant, termed *sample*, and A, B, C are the corresponding discrete-time system matrices.

The controller must finish its execution within a *deadline*, which is typically equal to the sampling period in case of the *logical execution time* (LET) paradigm [28]. Under the LET paradigm, the control input is computed based on the previously sampled state and applied exactly at the deadline. The dynamics of the discrete-time controller is as follows.

$$u[k] = K(x[k-1] - \gamma). \tag{2}$$

Here, K is the feedback gain matrix and γ is the plant output reference. In this work, we design the gain using the *linear quadratic regulator* (LQR) method [30] on *static controllers*, where a control action relies solely on the last sampled state.

a) Control Performance Requirements: Control implementation must ensure the underlying *performance requirements* of the system. We use the *settling time* [30] as one of the performance metrics. It is defined as the time required for the system output to reach and remain within a specified *output reference band*, i.e., the output reference $\pm$ a predefined small error margin (typically 1%–5% of the reference), even under sudden changes in the plant behavior (e.g., when a step input is applied). The plant exhibits a *transient response* while approaching the reference band after a step input, and reaches a *steady state* once it remains within this band. A shorter settling time indicates a faster responsiveness for a stable dynamical system, making it a key metric for evaluating plant performance. In hard real-time systems, meeting all deadlines together with a finite settling time ensures asymptotic stability.

To evaluate control performance, specifically in terms of control effort, we use the standard *LQR cost* function:

$$LQR\ cost = \sum_{k=0}^{\infty} \left(x^{\mathsf{T}}[k] Q x[k] + u^{\mathsf{T}}[k] R u[k] \right) \tag{3}$$

Here, $Q \succcurlyeq 0$ and $R \succ 0$ are symmetric weight matrices that penalize state deviation and control effort, respectively [6]. Minimizing the cost improves control performance.

Other performance metrics include *peak time*, which is defined as the time required to reach the first overshoot after a step input, and *rise time*, defined as the duration for the output to rise to 90% of the reference [30]. Smaller values for these metrics indicate a faster, more responsive system.

b) Real-Time Scheduling of a Control Task: A real-time scheduling of a control task provides a timed execution where every execution instance of the task must satisfy its deadline to ensure correct operation. A good scheduling strategy should also guarantee the desired control performance requirements. This work aims to find an optimal schedule that minimizes the LQR cost while satisfying the settling time requirement with other platform-level objectives and constraints.

B. Floating-Point Arithmetic

Floating-point numbers are the standard representation of real numbers in modern hardware due to their efficiency and widespread support. Their finite precision, however, introduces roundoff errors that propagate through computations and may amplify or dampen depending on the operations and input ranges. To model these errors, analyses typically follow the IEEE 754 standard [1]:

$$x \circ_{\mathrm{fl}} y = (x \circ y)(1 + e) + d, \qquad |e| \leq \varepsilon_m,\ |d| \leq \delta_m$$

where $\circ \in +, -, *, /$, and ε_m and δ_m denote relative and absolute error bounds for normal and subnormal values. Common formats include *half* (16-bit), *single* (32-bit), and *double* (64-bit) precision, with ε_m values of 2^{-11}, 2^{-24}, and 2^{-53}, and δ_m values of 2^{-24}, 2^{-150}, and 2^{-1075}. This model assumes rounding to the nearest and excludes overflow, which analyses usually prove cannot occur.

Worst-Case Error Analysis: Most roundoff error analyses bound the worst-case absolute error between real-valued computations $f(x)$ and their floating-point counterparts $\tilde{f}(\tilde{x})$ over bounded domains: $\max_{x \in [a,b]} |f(x) - \tilde{f}(\tilde{x})|$. Representative methods include dataflow analyses such as interval and affine arithmetic, used in tools like Daisy [12] and Fluctuat [15], and global optimization–based methods employed by FPTaylor [33], PRECiSA [29], and Satire [14].

III. Optimal Scheduling with Switching Precisions

In this section, we present our scheduling framework for optimizing per-sample precision-switching sequences. We start with a motivating running example from the automotive domain, use it to illustrate the formulation throughout, and then formulate the MIQP that synthesizes an optimal precision-switching sequence.

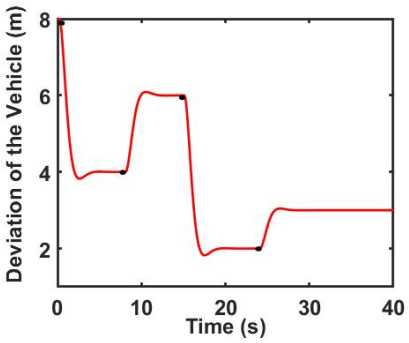

Fig. 1: Trajectory tracking control (TTC) system simulation

Running Example: We consider a second-order *trajectory tracking control* (TTC) system that adjusts the vehicle's acceleration to regulate its deviation from its desired trajectory. The discrete-time system matrices (see Section II-A) are:

$$A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix},\ B = \begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix},\ C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Let us assume that the controller can switch between two floating-point precisions: 16-bit and 32-bit, and we measure control performance using the LQR cost (lower is better). We simulate the system for $40\,\mathrm{s}$ (400 samples with sampling period $100\,\mathrm{ms}$) on an NVIDIA GPU that natively supports both formats. During the simulation, the vehicle's reference changes to $4\,\mathrm{m}$, $6\,\mathrm{m}$, $2\,\mathrm{m}$, and $3\,\mathrm{m}$ at $0\,\mathrm{s}$, $8\,\mathrm{s}$, $15\,\mathrm{s}$, and $24\,\mathrm{s}$, respectively (see Figure 1), to get the desired trajectory.

Executing the controller entirely in 16-bit yields an LQR cost of 22880. Executing entirely in 32-bit improves performance, achieving an LQR cost of 22166.8 (a 3.12% reduction), but increases runtime by 17.76% ($13.396\,\mathrm{ms}$ vs $11.017\,\mathrm{ms}$). This contrast highlights the key trade-off: higher precision improves control performance but increases execution time, whereas lower precision is faster but can degrade control performance.

Our Approach: Our goal is to exploit this precision trade-off by synthesizing a scheduling strategy that switches between floating-point precisions across samples, preserving the desired control performance while reducing execution time.

Given the controller configuration, system dynamics, and performance criteria, we compute an optimized *precision-switching sequence*, if one exists, that determines when different precisions should be used, such that, when implemented on the processor, both the total execution time of the controller and LQR cost are minimized while satisfying the settling time requirement. We formulate this scheduling as a multi-objective optimization problem that jointly optimizes control performance and total execution time, subject to the constraint that the system maintains settling time even under some step inputs applied in the reference over time.

Additionally, roundoff errors arising from floating-point arithmetic must be accounted for when computing the switching sequence. To ensure this, we introduce additional constraints in the optimization problem that incorporate the roundoff errors of the plant states, control inputs, and outputs.

type	variable	description
user-defined	A, B, C	system matrices
	K	controller gain
	Q, R	LQR weight matrices
	h	sampling period
	T_s	settling time
	γ	output reference
	δ	error margin around γ
	x_0, u_0	initial state and control input
	N	total samples
	$\mathcal{P}$	total precision options
	t_p	runtime per sample in p
	s_m	max switching overhead
	$\mathcal{I}$	switching intervals
precomputed	e_p	roundoff error in p
decision	$pr_{p,i}$	precision option for i
other	x_i, u_i, y_i	plant state, input, output at i

TABLE I: Variable notations (p: precision, i: sample)

.

However, directly encoding switching decisions and roundoff errors within the optimization problem introduces nonlinear constraints, which are typically non-convex and often intractable in practice. To make the problem tractable, we use sound over-approximations by performing linearizations where necessary and by precomputing sound bounds on the roundoff errors. The discrete-time LQR cost function used in the objective introduces quadratic terms, resulting in a Mixed-Integer Quadratic Program (MIQP). Despite being NP-hard in general, our problems remain convex and can be efficiently solved using state-of-the-art solvers [21], [24].

For `TTC`, our proposed approach synthesizes an optimal precision-switching sequence that executes the first 80 samples in 32-bit floating-point precision, the next 38 in 16-bit, the next 32 in 32-bit, and so on, with 32-bit used for the final 122 samples. The resulting LQR cost is 22259.4, a 2.71% reduction w.r.t. full 16-bit and only a 0.42% increase w.r.t. 32-bit execution. Moreover, the runtime is $11.572\,\mathrm{ms}$, yielding a 13.62% improvement over full 32-bit execution.

In the subsequent sections, we present the optimization problem formulation, describing each step in detail.

A. The MIQP Formulation

We first introduce the variables used in the MIQP problem in Table I, where variables specific to sample and precision are referred to with the subscripts i and p, respectively.

*a) **User-Defined Inputs**:* The plant dynamics matrices and the controller information (e.g., sampling period, LQR matrices, and controller gain) are provided by the user, as these are standard parameters of the controller design. The user also specifies the output reference and error margin (defining the band $\gamma \pm \delta$), the settling time T_s, the initial conditions, and the total number of samples N. The set of available precision options $\mathcal{P}$ is also generally known from the hardware.

Additionally, the worst-case per-sample runtime t_p for $p \in \mathcal{P}$, the maximum switching overhead s_m, and the switching intervals $\mathcal{I}$ are provided. Switching is permitted only within the intervals $[L_\beta, U_\beta] \in \mathcal{I}$, where each interval specifies a *range of sample indices* during which a precision change may occur. We use s_m as a conservative worst-case bound: the maximum time required to switch between any two precisions. The quantities t_p and s_m can be obtained via profiling or worst-case estimation; for $\mathcal{I}$, Section III-C1 presents one practical method to derive it (which we use in our experiments).

minimize: $z_1 + z_2$

where, $z_1 = \sum_{p=1}^{\mathcal{P}} \sum_{i=0}^{N} (pr_{p,i} \times t_p) + \sum_{\substack{i \in [L_\beta, U_\beta] \\ [L_\beta, U_\beta] \in \mathcal{I}}} s_m \times sw_i$

$z_2 = \sum_{i=0}^{N} (x_i^T \; Q \; x_i + u_i^T \; R \; u_i)$

subject to:

C : (**settling**) : $\gamma - \delta \leq \; y_i \; \leq \gamma + \delta, \quad i \geq \lceil \frac{T_s}{h} \rceil$

Fig. 2: The complete MIQP formulation

*b) **Precomputed Inputs**:* We precompute a sound bound on the roundoff error e_p for each precision option $p \in \mathcal{P}$ using the state-of-the-art error analysis tool [33], as described in Section III-B. These bounds are inputs to our optimization.

*c) **Decision Variables**:* The decision variable $pr_{p,i} = 1$ if precision p is used at sample i, and $pr_{p,i} = 0$ otherwise. For N samples and $\mathcal{P}$ precision options, the formulation includes $|\mathcal{P}| \times N$ binary decision variables.

*d) **Other Variables**:* The plant state x, control input u, and output y at sample i are computed under the precision selected by pr. The state and control updates (see Equations (1) and (2)) are performed for p-th precision when $pr_{p,i} = 1$.

1) Problem Formulation: We present the complete optimization formulation in Figure 2. Our formulation has two objectives: (1) *minimize* the total execution time with different precision executions (z_1), and (2) maximize control performance (z_2), which in this context corresponds to *minimizing* the LQR cost, as defined in Equation (3). Since both z_1 and z_2 are positive quantities, the combined objective, $z_1 + z_2$, jointly minimizes execution time and LQR cost.

The execution time objective z_1 consists of two components: the runtime of executing each sample in the chosen precision and the overhead incurred by precision switches. If sample i is executed in precision p (i.e., $pr_{p,i} = 1$), it contributes t_p to the total runtime; summing the terms $pr_{p,i} \times t_p$ over all samples i and precisions p yields the total execution time.

In addition, a switching overhead s_m is incurred whenever the selected precision changes between two consecutive samples. Switching is allowed only within the switching intervals $[L_\beta, U_\beta] \in \mathcal{I}$. We introduce a binary variable sw_i to indicate whether a switch occurs at sample i. Summing $s_m \times sw_i$ over all samples across all intervals then accounts for the total switching cost. This cost, together with the runtime term, yields the total execution time objective z_1 in Figure 2.

The optimization is constrained by the settling-time (C): the plant output y_i must reach and remain within the reference band $[\gamma \pm \delta]$ within $\lceil \frac{T_s}{h} \rceil$ samples (assuming an initial step input is applied to steer the system toward the reference).

$$
\begin{aligned}
&C_1: \sum_{p=1}^{\mathcal{P}} pr_{p,i} = 1,\\
&C_2: \sum_{L_\beta \le i \le U_\beta} (pr_{p,i-1} \oplus pr_{p,i}) \le 1,\\
&C_3: pr_{p,i} = pr_{p,U_\beta}, \quad i \in [U_\beta + 1, L_{\beta+1} - 1],\\
&C_4: pr_{p,i} = 1 \Rightarrow (x_i = x_i + e_p, \quad u_i = u_i + e_p, \quad y_i = y_i + e_p)
\end{aligned}
$$

Fig. 3: Non-linear switching constraints

2) Switching Constraints: The previous section introduced the binary variables $pr_{p,i}$ that select the precision $p \in \mathcal{P}$ at each sample i. This section formalizes their intended switching semantics through the additional constraints in Figure 3.

Constraint C_1 (unique precision per sample). For each sample i, exactly one precision is selected. Since $pr_{p,i} \in \{0,1\}$, the equality $\sum_{p=1}^{\mathcal{P}} pr_{p,i} = 1$ enforces that one variable is 1 and all others are 0, for all $1 \le i \le N$.

Constraint C_2 (at most one switch per interval). Each switching interval $[L_\beta, U_\beta] \in \mathcal{I}$ allows *at most one* change of precision (or none). Rather than enumerating all pairs of precisions (p, q) with $p \neq q$, we encode this property with an expression that scales linearly in $|\mathcal{P}|$. For each fixed precision p, the XOR term $pr_{p,i-1} \oplus pr_{p,i}$ is 1 exactly when the selection of p changes between samples $i-1$ and i (i.e., p is either entered or left). Summing this term over the sample indices in $[L_\beta, U_\beta]$ and constraining it by 1 ensures that such a change can occur at most once within the interval.

Recall our running example of `TTC` from Figure 1. If the first switching interval is $[L_1, U_1] = [38, 41]$, constraint C_2 is given by: $\sum_{38 \le i \le 41} (pr_{p,i-1} \oplus pr_{p,i}) \le 1$. The term $pr_{p,37} \oplus pr_{p,38}$ accounts for a switch at the start of the interval. If a switch occurs at $i = 38$, i.e., $pr_{p,37} \oplus pr_{p,38} = 1$, then all remaining terms for $39 \le i \le 41$ must be 0, so no further switch is allowed inside the interval.

Constraint C_3 (no switching outside intervals). Switches are restricted to the predefined switching intervals. Once an interval $[L_\beta, U_\beta]$ ends, the precision selection is held until the next interval $[U_\beta + 1, L_{\beta+1} - 1]$ begins for all samples. This constraint ensures that the precision remains unchanged on all samples between consecutive switching intervals.

For `TTC`, if $[L_1, U_1] = [38, 41]$ and the next interval starts at 80, then $pr_{p,i} = pr_{p,41}$, $\forall 42 \le i \le 79$ for every $p \in \mathcal{P}$.

Constraint C_4 (precision-dependent roundoff). As computations are carried out in the selected precision, the state, input, and output evaluations must account for the corresponding roundoff errors. Constraint C_4 adds the appropriate error term e_p whenever precision p is selected at sample i (i.e., $pr_{p,i} = 1$). This is enforced for all $1 \le i \le N$.

3) Linearization of Constraints: The switching constraints in Figure 3 make the formulation non-linear in two places: the XOR operator in C_2 and the implication in C_4. To obtain a tractable MIQP problem, we linearize both constructs exactly, without introducing any over-approximation. The resulting linear constraints are shown in Figure 4.

Linearizing the XOR in C_2. We replace each XOR term $pr_{p,i-1} \oplus pr_{p,i}$ with a new binary variable $\chi_{p,i}$. Constraints $C_2(1)$–$C_2(4)$ in Figure 4 enforce $\chi_{p,i} = pr_{p,i-1} \oplus pr_{p,i}$. For

$$
\begin{aligned}
&C_2(1): \chi_{p,i} \ge pr_{p,i-1} - pr_{p,i},\\
&C_2(2): \chi_{p,i} \ge pr_{p,i} - pr_{p,i-1},\\
&C_2(3): \chi_{p,i} \le 2 - (pr_{p,i-1} + pr_{p,i}),\\
&C_2(4): \chi_{p,i} \le (pr_{p,i-1} + pr_{p,i})\\
&C_2(5): \sum_{L_\beta \le i \le U_\beta} \chi_{p,i} \le 1\\
&\\
&C_4(1): e_i = \sum_{p=1}^{\mathcal{P}} e_p \times pr_{p,i}\\
&C_4(2): x_i = x_i + e_i\\
&C_4(3): u_i = u_i + e_i\\
&C_4(4): y_i = y_i + e_i
\end{aligned}
$$

Fig. 4: MIQP with linear constraints

example, $(pr_{p,i-1}, pr_{p,i}) = (0,1) \Rightarrow \chi_{p,i} = 1$, while $(0,0)$ or $(1,1) \Rightarrow \chi_{p,i} = 0$. Constraint $C_2(5)$ then replaces the original switching-count constraint by summing $\chi_{p,i}$ over all samples in the switching interval.

Linearizing the implication in C_4. We introduce an auxiliary variable e_i to capture the roundoff error associated with the precision selected at sample i. Constraint $C_4(1)$ links e_i to the switching variables so that it picks the corresponding precomputed bound e_p. Constraints $C_4(2)$–$C_4(4)$ then add e_i to the state, input, and output evaluations, ensuring that the roundoff error matches the selected precision at each sample.

These additional constraints thus eliminate all non-linearity introduced by C_2 and C_4, yielding an MIQP with completely linear constraints.

4) Linearizing Switching Overhead in the Objective: In addition to the non-linear constraints handled above, the execution-time objective z_1 considers a switching-overhead term, including the switching variable sw (see Figure 2). Using the XOR indicators $\chi_{p,i}$ from Section III-A3, this overhead can be expressed exactly with a linear term:

$$
z_{12} = \sum_{[L_\beta, U_\beta] \in \mathcal{I}} \left(\tfrac{s_m}{2} \times \sum_{p=1}^{\mathcal{P}} \left(\sum_{i \in [L_\beta, U_\beta]} \chi_{p,i} \right) \right). \tag{4}
$$

Recall that $\chi_{p,i} = 1$ iff the selection of precision p changes between samples $i-1$ and i. If a switch occurs at sample i within $[L_\beta, U_\beta]$, then exactly two precision variables change status at that sample (one is left and one is entered), implying

$$
\sum_{p=1}^{\mathcal{P}} \sum_{i \in [L_\beta, U_\beta]} \chi_{p,i} = \begin{cases} 2, & \text{if a switch occurs in } [L_\beta, U_\beta], \\ 0, & \text{otherwise.} \end{cases}
$$

Consequently, the expression inside the outer summand in Equation (4) evaluates to s_m if a switch occurs in the interval, and to 0 otherwise. Summing over all β (i.e., accounting for all intervals) yields the total switching cost z_{12}. This exactly linearizes the switching overhead in z_1.

Unlike z_1, the LQR term z_2 is inherently quadratic and cannot be linearized exactly without changing the objective. In this work, we apply exact linearizations where possible (e.g., for the XOR and implication constructs) and introduce approximation only through conservative roundoff-error bounds. We therefore retain z_2 in its original quadratic form to avoid introducing additional over-approximation. In principle, one could

replace z_2 with a linear relaxation or over-approximation, but this would optimize a surrogate objective and can distort the runtime performance trade-off by over-penalizing low-precision samples, potentially biasing the solution toward selecting high precision most of the time (or even always).

The resulting formulation is an MIQP with a linear execution-time objective (z_1, including z_{12}), a convex quadratic performance objective (z_2), and entirely linear constraints. This MIQP class is efficiently solved by state-of-the-art solvers such as Gurobi and CPLEX [21], [24], and our experiments in Section IV-2 also confirm that the quadratic is not a bottleneck.

Modeling Choices: Our MIQP formulation uses a few conservative modeling choices. First, it uses a scalar roundoff-error bound e_p for each precision p, which keeps the formulation compact but may over-approximate the actual error and yield fewer low-precision samples than a more precise error model; this is discussed in Section III-B. Second, the switching overhead s_m is modeled as a constant worst-case value. If more detailed platform-specific overheads are known, they can be incorporated into the execution-time objective. Finally, the formulation assumes a given set of switching intervals $\mathcal{I}$; overly restrictive intervals may limit useful switching opportunities. Nevertheless, all these choices ensure that the synthesized sequence satisfies the required control performance constraints while still providing useful precision switching that optimizes execution time.

B. Computing Roundoff Errors

The previous sections assume the roundoff error bounds as precomputed inputs to the optimization problem. This section describes how we obtain these bounds.

We compute sound worst-case roundoff errors using FPTaylor [33], a state-of-the-art tool for floating-point accuracy analysis. Given bounds on input variables, FPTaylor soundly maximizes the corresponding roundoff error expressions using global optimization techniques and returns mathematically rigorous upper bounds.

In our setting, FPTaylor requires ranges for the plant state x and control input u. These ranges correspond to physically meaningful quantities and are typically available from the plant-control model, design specifications, and/or simulation traces. Accordingly, we assume such bounds are known or can be conservatively estimated during controller design. Using these ranges, we compute a roundoff bound e_p for each available precision $p \in \mathcal{P}$, and supply $\{e_p\}_{p\in\mathcal{P}}$ to the optimization.

The scalar bound e_p on roundoff errors is a *conservative abstraction* of the worst-case error in precision p over the given input ranges. The soundness of this bound follows from [33]. Since the ranges conservatively cover all possible values of x and u across all samples, e_p safely bounds the possible roundoff errors throughout the computation. The actual error may be smaller for a particular sample or dimension. Our formulation applies this worst-case bound at each sample, and the resulting updated state is propagated through the closed-loop iteration. Thus, cross-sample effects are accounted for conservatively during optimization.

For the running example of `TTC`, the system has two state variables and one control input. We obtain the bounds on these variables directly from the control model, design specifications, and through simulations. For 16-bit and 32-bit floating-point precision levels, the roundoff error bounds are computed as 1.01×10^{-2} and 1.21×10^{-6} respectively.

While this paper focuses on floating-point controller implementations, the proposed technique is generic and can also be applied to fixed-point controllers. For that setting, we assume that candidate fixed-point implementations are available before solving the MIQP, e.g., formats with specific bit-widths and scaling choices obtained from the designer, the target hardware, or a separate fixed-point analysis/synthesis step. A key additional consideration is that fixed-point arithmetic allows many bit-width choices; if a large set of formats is considered, the number of precision options $|\mathcal{P}|$ (and thus the number of decision variables and constraints) grows, increasing the size of the MIQP. This growth is linear in $|\mathcal{P}|$, and modern MIQP solvers can handle large numbers of variables and constraints efficiently. Moreover, in practice, only a limited set of fixed-point formats is typically supported by the target hardware or considered by the designer, making our approach viable in the fixed-point setting as well. Finally, fixed-point roundoff bounds will require a different round-off error analysis: while we use FPTaylor for floating-point error computation, one can instead use tools such as Daisy [12] to compute sound error bounds for fixed-point implementations. Thus, once candidate fixed-point implementations are characterized by their runtime and sound error bounds, they can be incorporated into our framework in the same way as floating-point options.

C. From Theory to Practice

We now describe how the proposed formulation can be instantiated in practice. We first present the switching-interval construction we use for our experiments, and then describe the automated tool pipeline and the choice of backends for MIQP solving and roundoff-error analysis.

1) Designing Switching Intervals: Ideally, the set of switching intervals, $\mathcal{I}$, is an user-provided input; the proposed optimization supports any given $\mathcal{I}$. For our implementation and experiments, we construct $\mathcal{I}$ automatically from standard time-domain performance metrics, namely, rise time T_r, peak time T_p, and settling time T_s, to focus switching opportunities on phases where precision most strongly affects control performance. This interval-based policy also enforces a minimum spacing between switches, preventing overly frequent switching in practice.

The time-domain metrics used to construct $\mathcal{I}$ are typically available from the control-design phase. When they are not explicitly given, they can often be estimated from the closed-loop poles, damping ratio, and natural frequency [30]. For higher-order systems, if these quantities are not directly computable, they can be obtained through simulation or numerical approximation. Therefore, it is reasonable to assume that the required time-domain metrics are available or can be obtained for the class of LTI systems considered in this work.

a) Motivation: Consider a control system with a settling time $T_s = 1\,\text{s}$, output reference $\gamma = 0.5$, and an error margin of 5%. The output is expected to reach and remain within $[0.45, 0.55]$ within $1\,\text{s}$. In low-precision arithmetic, accumulated roundoff may delay convergence (e.g., reaching only 0.422 at $1\,\text{s}$), violating the settling-time requirement and prolonging deviations from the reference. Since shorter settling, peak, and rise times indicate fast response and better performance (see Section II), it is desirable to adapt the controller precision accordingly. Hence, we construct the switching intervals based on these metrics.

We assume the system is driven toward its reference γ by an initial step input (see Section III-A1). For our implementation and experiments, we allow additional reference changes by applying step inputs over time. This setup allows us to evaluate how effectively the optimization decides precision switches. Particularly, how the switching decisions promote fast recovery from transient responses while balancing runtime and control performance. Let us consider that r step inputs are applied in time instances $\{t_j\}_{j=1}^{r}$, and their corresponding desired output references are $\{\gamma_j\}_{j=1}^{r}$.

b) Interval Construction: Immediately after a step input at t_j, the system exhibits its largest deviation from the reference γ_j indicating its transient response: during the initial rise (until approximately 50% of T_r) and near the peak overshoot before it settles. These phases are precisely where higher numerical precision tends to be most influential, as it helps satisfy performance requirements and accelerates convergence.

Thus, for each step input applied at t_j $(2 \leq j \leq r)$, we construct two switching intervals:

1) **Early transient (rise)** interval that covers the period from the step input instant to half of the rise time:

$$[L_{2j-2}, U_{2j-2}] = \left[\left\lceil \frac{t_j}{h} \right\rceil, \left\lceil \frac{t_j + T_r/2}{h} \right\rceil\right].$$

2) **Peak-to-settle** interval that covers the time from the first peak overshoot to when the system settles:

$$[L_{2j-1}, U_{2j-1}] = \left[\left\lceil \frac{t_j + T_p}{h} \right\rceil, \left\lceil \frac{t_j + T_s}{h} \right\rceil\right].$$

The system starts with an initial step input applied at $t_1 = 0$. Hence, the first interval corresponds to the initial transient response and is defined as:

$$[L_1, U_1] = \left[\left\lceil \frac{T_p}{h} \right\rceil, \left\lceil \frac{T_s}{h} \right\rceil\right].$$

This choice conservatively runs the initial samples (up to $\lceil T_p/h \rceil$) in the highest available precision, since the system is far from steady state, and permits the first precision switch only at sample $\lceil T_p/h \rceil$.

As our proposed switching approach is interval-based (i.e., at most one switch per interval $[L_\beta, U_\beta]$)), an early switch to low precision after the initial step input can force a long low-precision segment. The resulting roundoff error accumulation may prevent the output from reaching the reference band. Therefore, we execute the first initial transient in the highest precision until $\lceil T_p/h \rceil$; the optimization then decides whether and when (or not) to switch thereafter, thus maximizing control performance under the settling time constraint.

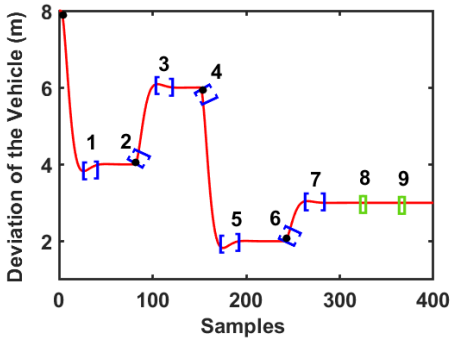

Fig. 5: Switching intervals for the TTC system

We illustrate the interval construction using our running example TTC. The system has settling time $T_s = 4.1\,\text{s}$, peak time $T_p = 3.8\,\text{s}$, and rise time $T_r = 2\,\text{s}$, with sampling period $h = 100\,\text{ms}$. Let us also assume that four step inputs are applied at $0\,\text{s}$, $8\,\text{s}$, $15\,\text{s}$, and $24\,\text{s}$ near the desired references of $4\,\text{m}$, $6\,\text{m}$, $2\,\text{m}$, and $3\,\text{m}$, respectively. Applying the construction above yields the following switching intervals covering a range of samples (as shown in blue in Figure 5):

$1)[L_1, U_1] = [38, 41]$ (for $t_1 = 0s$),
$2)[L_2, U_2] = [80, 90]$, $3)[L_3, U_3] = [118, 121]$ (for $t_2 = 8s$),
$4)[L_4, U_4] = [150, 160]$, $5)[L_5, U_5] = [188, 191]$ (for $t_3 = 15s$),
$6)[L_6, U_6] = [240, 250]$, $7)[L_7, U_7] = [278, 281]$ (for $t_4 = 24s$).

c) Additional Switching: While the intervals above target transient phases, it can also be beneficial to allow limited switching after the system has settled (at $\lceil (t_r + T_s)/h \rceil$), e.g., to trade small performance gains for runtime reductions (or vice versa). At the same time, allowing frequent switching can be counterproductive: it incurs repeated overhead and may amplify numerical transients due to repeated precision casting; therefore, post-settling switching opportunities should be introduced sparingly.

In TTC, the last step input occurs at $t_r = 24\,\text{s}$, i.e., at the 240-th sample, and the output settles approximately by $t_r + T_s$, i.e., by $28.1\,\text{s}$ (or 281 samples). The simulation continues until $40\,\text{s}$ (400 samples), during which the output remains within the reference band. Executing the remainder entirely in high precision would be wasteful, but using only low precision could still degrade performance (e.g., near steady state, higher precision may matter for fine adjustments). We therefore permit additional switching after settling to balance runtime and control performance while avoiding excessive switching, which may incur unnecessary switching overhead.

To model this, we introduce periodic switching opportunities every $\lceil T_s/h \rceil$ samples after the system has remained settled for additional $\lceil T_s/h \rceil$ samples, starting at $\tau = \lceil (t_r + 2T_s)/h \rceil$. Concretely, we add a switching interval of one-sample every $\lceil T_s/h \rceil$ samples thereafter:

$$[\tau + k\lceil T_s/h \rceil,\ \tau + k\lceil T_s/h \rceil + 1], \quad k = 0, 1, 2, \ldots$$

until N samples are reached. Hence, the total number of switching intervals for r step inputs is: $(2r - 1) +$

$\left\lceil \frac{N-\lceil (t_r+2T_s)/h \rceil}{\lceil T_s/h \rceil} \right\rceil$.

For TTC, this yields a total of 9 intervals, including the additional switching intervals as $[L_8, U_8] = [322, 323]$ and $[L_9, U_9] = [363, 364]$ (shown in green in Figure 5).

2) Implementation: We have implemented our approach in a prototype tool called *Apex*, available on GitHub: https://github.com/DBanerjee19/Apex. Apex takes as input user-defined parameters (see Table I) together with system timing properties (rise, peak, and settling times) and outputs the *optimal precision-switching sequence*. Then it automatically (i) constructs switching intervals ($\mathcal{I}$) from the timing properties using the interval-construction described above, (ii) generates the corresponding MIQP instance, (iii) invokes a state-of-the-art MIQP solver, and (iv) returns the optimal switching sequence. Roundoff errors are computed by profiling ranges for the state variables and control inputs and supplying them to an off-the-shelf sound roundoff error analyzer. This step is currently performed manually, but full integration into the tool can be straightforwardly implemented.

3) Choice of External Tools: We use Gurobi [21] as the MIQP solver. Gurobi is a high-performance industrial solver that efficiently handles convex MIQP problems with quadratic objective function and linear constraints, which aligns with our formulation.

Gurobi performs computations in 64-bit floating point and relies on feasibility and optimality tolerances typically around 10^{-9}. While our formulation is agnostic to the available precision options, these tolerances effectively limit the current implementation to at most 32-bit floating-point precisions. At higher precisions (e.g., 64-bit), the roundoff errors can become comparable to, or smaller than, the solver tolerances, potentially leading to unreliable optimization behavior. Other commercial solvers such as CPLEX [24] use similar numerical tolerances and therefore face comparable limitations. The SCIP Optimization Suite [16] supports tighter tolerances (reported up to 10^{-15}), improving numerical robustness but typically at a significant performance cost and still not guaranteeing reliable separation of 64-bit roundoff errors. Nevertheless, 32-bit floating point is common in embedded control applications, making this choice realistic and practically sufficient.

For *roundoff errors*, we use FPTaylor [33], a global optimization–based tool that computes sound worst-case error bounds, often more precise than other alternative approaches. In principle, however, any tool that computes sound worst-case roundoffs for the target arithmetic can be used.

IV. Evaluation

This section evaluates our approach, and the Apex tool considering two available floating-point precision options ($|\mathcal{P}| = 2$): 16-bit (*FP16*) and 32-bit (*FP32*). While our formulation supports arbitrary $\mathcal{P}$, sub-16-bit floating-point formats (e.g., 8-bit) were not supported on the CPU/GPU platforms available to us, and precisions above FP32 provide roundoff errors that are often below Gurobi's tolerances. FP16/FP32, therefore, represents the most practical and widely supported design space for evaluating precision switching.

We note that the runtime benefit of precision switching depends on the availability of efficient native hardware support for the selected precisions. Platforms with native support for both FP16 and FP32 are increasingly available, including newer Arm Cortex-M cores such as Cortex-M52 [3], Cortex-M55 [4], and Cortex-M85 [5] when equipped with the appropriate floating-point unit. These cores are used in cyber-physical systems, including industrial control systems and robotics, making FP16/FP32 a realistic precision space for such deployments. On platforms where lower precision is not natively supported and must instead be emulated in software, the execution-time and switching-overhead tradeoffs may change significantly, and precision switching may offer limited runtime benefit. At the same time, our formulation accounts for this platform dependence by using execution times (including software emulation overhead) and switching overheads as inputs. Thus, if such overheads are known in advance, they can be incorporated into the optimization, and the optimizer will avoid precision switching when it is not beneficial for the target hardware.

We consider six execution modes in total. Five of these serve as baselines: two uniform-precision executions, (1) all-FP16 and (2) all-FP32, and three heuristic switching executions, (3) **ITS** (*initial transient switching*), where the controller uses FP32 during the initial transient after every step change and FP16 afterward; (4) **PS** (*periodic switching*), where the controller alternates between FP16 and FP32 every 10 samples; and (5) **RS** (*random switching*), where the controller randomly selects between FP16 and FP32 at each switching point. The remaining mode is (6) optimized switching execution (**OPT**), where the controller switches between FP16 and FP32 according to the sequence generated by our optimization.

We evaluate our approach using the following research questions:

RQ1. How do control performance and runtime vary across precision options (FP32 vs. FP16)?
RQ2. For a given control system, how does the optimized switching sequence perform?
RQ3. How does the optimized switching sequence compare to the baselines w.r.t. runtime and control performance?
RQ4. Can precision-switching sequences be useful in multi-task scheduling on a shared processor?

1) Benchmarks: We use 11 benchmark systems from the automotive domain, with system orders (no. of state variables) ranging from 2 to 5. Each benchmark models a feedback control system, summarized below.

We consider 6 **second-order** systems: 1) the *DC-motor speed controller* (MS) [31] regulates motor speed by adjusting terminal voltage; 2) the *F1-Tenth lane-following controller* (F1) [22] keeps a small autonomous car aligned with the lane center by controlling its steering angle; 3) the *resistor–capacitor circuit* (RC) [22] models voltage regulation for signal filtering; 4) the *vehicle dynamics controller* (VDC) [2] prevents skidding by regulating the vehicle's yaw rate through steering adjustments; 5) the *trajectory tracking controller* (TTC) [2] controls the deviation of a vehicle from its reference trajectory by adjusting its acceleration; and 6) the *DC-servo controller* (DCS) [11] adjusts servo acceleration to regulate angular position.

The 2 **third-order** systems are: 1) the *cruise controller*

(CC) [31], which maintains vehicle speed by modulating throttle force and 2) the *adaptive cruise controller* (ACC) [11], which regulates spacing error with respect to a leading vehicle.

The 2 **fourth-order** systems include: 1) the *lane-keeping controller* (LK) [11], that minimizes lateral position error by computing appropriate steering inputs and 2) the *suspension controller* (SC) [31], which stabilizes the vehicle's vertical position by adjusting input forces.

Finally, we consider a **fifth-order** system, the *vision-based lateral controller* (LC) [17], that uses image-based feedback to compute the front-wheel steering angle required to maintain the vehicle's position within the lane.

The ACC and CC are *open-loop unstable* systems. They become completely unstable without the control input from their feedback controllers [30]. The remaining ones are *open-loop marginally stable*, operating around an unstable equilibrium. Without the control input, they neither converge to the reference nor diverge, but sustain oscillations [30]. In all cases, the closed-loop controller stabilizes the system.

2) Experimental Setup: All experiments for computing optimal switching sequences were conducted on a 64-bit Windows OS with an Intel Core i3 processor (1.20 GHz) and 8 GB RAM. We use Gurobi v12.0.2 to solve the MIQP instances and FPTaylor v0.9.3 for sound roundoff error analysis. In the error analysis, we account for roundoff errors in input and use the rounding mode corresponding to the precision used.

To compare all six execution modes, we implemented separate GPU simulations for FP16, FP32, OPT, ITS, PS, and RS. We run these simulations on an NVIDIA GPU with 24 GB of RAM and Tensor Core support, providing native hardware support for FP16 and FP32 arithmetic. To ensure a fair comparison across execution modes, we disabled compiler-level auto-optimization flags.

RQ1: FP16 vs FP32

We begin by comparing the two uniform precision options: executing the controller entirely in FP16 and entirely in FP32. This comparison establishes the trade-off between the runtime of lower precision and its potential impact on control performance.

We evaluate control performance using the *LQR cost* (lower is better). Table II reports the results. Columns 1 and 2 present all the benchmarks with their sampling periods (h) and settling times (T_s), and the minimum and maximum values of output references, respectively. Column 5 presents the percentage reduction in LQR cost when using FP32 instead of FP16 (positive values indicate that FP32 improves performance).

Control Performance: Across *all* benchmarks, FP16 yields higher LQR costs than FP32, indicating degraded control performance. The effect is particularly severe for the open-loop unstable benchmarks CC and ACC, where FP16 produces *NaN* or *Inf* costs, consistent with numerical overflow and error amplification under unstable dynamics. In contrast, FP32 yields finite costs for all benchmarks.

We also observe that the *scale* of the LQR cost varies substantially across systems. Unstable systems (CC, ACC) exhibit large FP32 costs because their dynamics demand larger control efforts to reach and maintain the desired reference. Moreover, ACC incurs a higher cost than CC, because of its larger control effort, stemming from a longer sampling period, i.e., a slower sampling rate (see Column 1 of Table II). Among the marginally stable systems, costs are generally lower, although DCS is an exception: FP16 yields *Inf* and FP32 yields a large but finite cost, indicating that even marginal stability can be numerically fragile under low precision, where rounding and overflow effects accumulate.

Note that absolute LQR costs are not directly comparable across benchmarks, as the cost also depends on the magnitude of the system output variable and other state variables. For example, F1 system, the (min, max) values of the reference output are $(0.1, 0.4)$ and the LQR cost obtained is 30.97, whereas the cost is 419.68 for RC with the (min, max) as $(0.0, 5.0)$. We therefore interpret Table II primarily via the FP16–FP32 gap within each benchmark, which reflects the system's sensitivity to numerical precision.

Overall, FP32 consistently yields finite and substantially lower costs in all benchmarks, regardless of whether the system is open-loop unstable or marginally stable, with the largest improvement of 82.50% for VDC and the smallest of 3.22% for F1, indicating superior control performance.

Runtime: We next compare FP16 and FP32 in terms of total runtime on the GPU. For each benchmark and precision, we measure (i) the *per-sample worst-case runtime* t_p (the maximum observed runtime of a single closed-loop iteration

benchmark $(h(ms), T_s(s))$	ref. output (min, max)	LQR cost		
		FP16	FP32	red%
MS (20, 0.6)	(0.00, 1.50)	856.00	572.33	33.14
F1 (20, 0.6)	(0.10, 0.40)	32.00	30.97	3.22
RC (20, 0.3)	(0.00, 5.00)	669.50	419.68	37.31
VDC (100, 0.8)	(0.00, 1.00)	$0.0^{\star}$	$0.0^{\star}$	82.50
TTC (100, 4.1)	(2.00, 8.00)	22880.00	22166.80	3.12
DCS (10, 0.8)	(4.00, 10.00)	*Inf*	2561960.0	—
CC (10, 1.2)	(15.00, 60.00)	*NaN*	341106.0	—
ACC (20, 0.8)	(1.00, 7.00)	*Inf*	575550.0	—
LK (20, 0.8)	(0.03, 1.00)	48.09	42.54	11.54
SC (20, 0.4)	(0.03, 0.50)	58.53	56.45	3.55
LC (10, 0.8)	(-0.08, 0.08)	29.31	28.13	4.03

TABLE II: FP16 vs FP32 in terms of LQR costs
($*$ VDC: cost $= 4 \times 10^{-3}$ for FP16, $= 7 \times 10^{-4}$ for FP32)

	runtime					
	per-sample worst-case (ms)			total ex. time (ms)		
	FP32	FP16	red. %	FP32	FP16	red. %
MS	0.187	0.149	20.321	4.517	4.260	5.690
F1	0.175	0.146	16.571	11.899	11.150	6.295
RC	0.205	0.161	21.463	4.736	4.222	10.853
VDC	0.189	0.142	24.868	6.499	5.916	8.971
TTC	0.214	0.147	31.308	13.396	11.017	17.759
DCS	0.211	0.143	32.227	23.174	21.892	5.532
CC	0.182	0.146	19.780	22.822	22.041	3.422
ACC	0.180	0.154	14.444	9.351	8.253	11.742
LK	0.161	0.146	9.317	11.662	11.147	4.416
SC	0.190	0.145	23.684	12.055	11.095	7.963
LC	0.268	0.155	42.164	23.031	22.168	3.747

TABLE III: FP16 vs FP32 in terms of runtime

over three runs), and (ii) the *total execution time* of executing all samples (averaged over three runs). Table III summarizes the results. Columns 4 and 7 report percentage reductions for FP16 relative to FP32.

As expected, FP16 is faster for *all* benchmarks: averaged across systems, FP16 reduces the per-sample observed worst-case runtime by 23.286% and the total execution time by 7.854% compared to FP32. The smaller improvement in total time is expected, as end-to-end execution includes fixed costs (e.g., kernel launches and memory transfers) that are less sensitive to arithmetic precision than the per-sample compute kernel itself. Total runtimes also vary with system order and simulation length: e.g., LC (fifth order, 800 samples) has a much larger total time than VDC (second order, 200 samples).

These baselines shows the core trade-off: FP32 provides consistently better (and often necessary) control performance, whereas FP16 offers substantial runtime savings. This motivates precision switching: for many benchmarks, selectively using FP32 only when needed can approach FP32-level performance while retaining part of the FP16 runtime benefit. At the same time, benchmarks with extreme FP16 fragility (e.g., CC, ACC, DCS) indicate that using low precision for even a short period of time can be difficult.

RQ2: Optimal Precision Switching

We now evaluate the effectiveness of our MIQP-based precision-switching approach, through the optimal sequences OPT it generates for all benchmarks.

Each system's parameters, such as sampling period, settling time, output reference band, are configured according to its control design. Whenever available, we use plant state and control input ranges provided by the model or design specifications; otherwise, we estimate feasible ranges from simulation traces. While the simulated bounds are not sound, they are conditioned on realistic operational ranges and provide sufficiently tight estimates for our evaluation. This assumption is also practical, as control engineers typically know or can conservatively bound such ranges during controller design. We use these ranges to compute the roundoff errors. Together with the system dynamics and the measured per-sample worst-case runtimes t_p (Columns 2–3 of Table III), these quantities form the inputs to the optimization problem. (All experimental data are provided with the tool.)

Columns 2–7 in Table IV report the precomputed inputs to our optimization problem and its results. Columns 2, 3, and 4 present e_p for FP16 and FP32 and the total number of switching intervals (#intvl) produced by our interval-construction (Section III-C1). Columns 5–6 show the synthesized sequence: the number of precision-switches (#sw) and the percentage of FP16 samples. Column 7 reports *end-to-end optimization time* (formulation + solving), averaged over three runs.

The optimizer chooses *no* FP16 samples for DCS, CC, and ACC. This is consistent with the baseline results in Table II: these systems exhibit severe FP16 fragility (FP16 yields *Inf*/*NaN*), so even executing a few FP16 samples can degrade control performance. Note that these systems also have comparatively large FP16 roundoff errors in Table IV, reinforcing that roundoff error magnitude is a useful indicator of FP16 viability, although it is not sufficient on its own.

For the remaining 8 benchmarks, the synthesized schedules use both precisions, often with a substantial fraction of FP16 samples. Three systems (F1, RC, VDC) run predominantly in FP16 ($85 - 97\%$), indicating that low precision can be used safely for most samples when the FP16 error bound is small relative to the control task's tolerance. In contrast, SC uses only 10.5% FP16, aligning with its relatively large FP16 error bound; here, the optimizer reserves FP32 for most samples to satisfy the settling time requirement.

The number of switches varies considerably across systems, even when #intvl is large. For example, VDC performs only 1 switch out of 24 intervals while executing 97% of samples in FP16, indicating that frequent switching is unnecessary. This aligns with VDC's small control effort (near-zero LQR cost in Columns 3–4 of Table II), allowing extended low-precision execution without violating the settling time constraint. In contrast, TTC and LK use multiple switches (6 each), suggesting that they exhibit more pronounced precision-sensitive phases in which switching between FP16 and FP32 is beneficial.

Column 7 in Table IV shows that our approach is also efficient. For 10 out of 11 benchmarks, end-to-end optimization completes in under 3.1 s; LC takes 12.48 s, consistent with its larger problem size (higher-order system). The optimization time scales with the number of samples, the number of switching intervals, and the resulting MIQP size. Nonetheless, it remains practical overall, indicating that solving MIQP with Gurobi is not a bottleneck for generating switching sequences.

After generating the optimized switching sequences, we simulate each sequence on the GPU. The resultant LQR cost and runtime are presented in Columns 8–9 of Table IV. Benchmarks for which no optimized switching sequence is obtained are marked with '-'. We report the switching overhead, i.e., the runtime spent on variable casts/truncations during precision switching in Column 10. These results show that the switching overhead is negligible across all benchmarks, remaining below 0.0018 ms, compared to total execution times that are typically above 4.3 ms. This corresponds to less than 0.04% of the total execution time, suggesting that precision switching is a viable option in practice.

RQ3: Our Approach vs Baselines

We compare the performance of our generated precision-switching sequences (OPT) against five baselines: two uniform-precision executions, all-FP16 and all-FP32, and three heuristic switching strategies, ITS (initial transient switching), PS (periodic switching), and RS (random switching).

For the CC, ACC, and DCS systems, OPT selects no FP16 samples. Interestingly, in these systems, all three heuristic switching strategies exhibit the same behavior as all-FP16, yielding *NaN* LQR cost for CC and *Inf* LQR costs for ACC and DCS. This confirms that OPT correctly avoids FP16 in these cases. Since the heuristic baselines do not produce meaningful finite LQR costs for these benchmarks and OPT does not perform precision switching, we exclude them from the following comparison.

	optimization results						simulation results		
	precomputed inputs			outputs		time (s)	LQR cost	runtime	
	roundoff (e)		#intvl	#sw	% FP16 samples			total ex. time (ms)	switch ovhd (ms)
	FP16	FP32							
MS	2.27e-3	2.51e-7	4	2	33.33	0.31	572.002	4.30	0.0005
F1	7.58e-4	7.86e-8	9	5	85.50	3.10	31.813	11.37	0.0009
RC	9.13e-3	1.08e-6	9	2	86.00	0.34	419.750	4.56	0.0006
VDC	7.37e-4	6.55e-8	24	1	97.00	0.22	0.0^*	6.17	0.0003
TTC	1.01e-2	1.21e-6	9	6	28.75	1.64	22259.4	11.57	0.0017
DCS	5.68e-2	7.21e-6	10	0	0.00	2.55	—	—	—
CC	11.62e-1	1.36e-4	8	0	0.00	2.30	—	—	—
ACC	1.41e-1	1.49e-5	7	0	0.00	0.93	—	—	—
LK	2.34e-2	2.97e-6	9	6	26.25	2.88	42.429	11.15	0.0015
SC	1.62e-1	1.85e-5	13	6	10.50	2.55	56.484	11.18	0.0016
LC	4.95e-4	3.71e-8	7	6	29.25	12.48	28.601	22.33	0.0016

TABLE IV: Optimization results – precomputed error bounds, synthesized switching sequences, and optimization time; Simulation results – LQR cost and runtime of the optimized precision-switching sequences ($*$ VDC: cost $= 7 \times 10^{-4}$)

	LQR cost				
	% reduction w.r.t.				
	FP32	FP16	ITS	PS	RS
MS	0.057	33.177	0.000	0.000	0.087
F1	−2.722	0.584	0.049	−1.394	−0.842
RC	−0.017	37.304	0.060	0.000	0.178
VDC	0.000	82.500	0.011	0.011	−0.056
TTC	−0.418	27.124	1.889	0.981	1.021
LK	0.261	11.772	0.021	0.646	−0.605
SC	−0.060	3.496	0.028	−0.802	−0.012
LC	−1.674	2.419	2.218	0.518	0.657

TABLE V: Relative change in LQR cost of OPT with respect to baselines. Negative values indicate a higher cost for OPT.

Control Performance: Table V compares the control performance of OPT with the baseline configurations in terms of LQR cost. Columns 2–6 report the percentage change of OPT relative to FP32, FP16, ITS, PS, and RS, respectively. Negative values indicate that OPT incurs a higher LQR cost than the corresponding baseline.

Across *all* benchmarks, the LQR costs in OPT remain close to FP32, with changes typically below 2.722% (average 0.815%), indicating near-FP32 control performance. Since FP32 is expected to provide the best performance, OPT usually incurs a slightly higher LQR cost, reflected by negative values in Column 2. The only exceptions are MS and LK, where OPT achieves marginally lower costs than FP32. Such small improvements can occur when rounding effects partially cancel across the computation.

At the same time, optimized switching sequences substantially improve over all-FP16 execution: reductions range from 0.584% (F1) to 82.50% (VDC), with an average reduction of 24.797%. This behavior aligns with the structure of the optimized schedules: FP32 is assigned to precision-sensitive phases, while FP16 is used when the output is safely within (or close to) the reference band, limiting the accumulation of roundoff error.

ITS performs better than all-FP16 execution because it uses FP32 during the sensitive initial transient phase after a step input. However, ITS then continues with FP16 for the remaining samples, whereas OPT can introduce additional switches during both transient and near-steady-state phases, as described in Section III-C1. As a result, OPT improves over ITS in most cases, with an average LQR-cost reduction of 0.535%. The two approaches behave identically for MS and show only a small improvement for VDC, where the optimized schedules involve only a limited number of step inputs and switches (see Column 5, Table IV).

The other strategies, PS and RS, do not explicitly account for transient or near-steady-state behavior. As a result, their control performance is less consistent. In some systems, such as F1 and SC, both PS and RS achieve lower LQR costs than OPT; RS also performs slightly better for VDC and LK. However, OPT improves over PS in four out of eight benchmarks, with two additional cases showing identical LQR costs, and improves over RS in four out of eight benchmarks. The largest reductions are 0.981% for PS on TTC and 1.021% for RS on TTC.

These results are expected: our optimization jointly optimizes control performance and execution time, so it may accept a slightly higher LQR cost when the runtime savings justify it. Simple heuristics like PS and RS can therefore match or outperform OPT on LQR cost alone for some benchmarks, but their performance is instance-dependent.

In summary, OPT achieves near-FP32 control performance, consistently improves over all-FP16, and provides a more principled way to realize the desired performance-runtime trade-off than simple heuristics.

Runtime: We next compare the total execution time on the GPU, averaged over three runs. Columns 2–6 of Table VI report the percentage runtime reduction of OPT relative to FP32, FP16, ITS, PS, and RS, respectively. Negative values indicate that OPT is slower than the corresponding baseline.

As expected, all-FP16 achieves the lowest runtime among all baselines. The runtime of OPT increases by only 2.719% relative to FP16, while it is 5.794% faster than FP32. Thus, OPT achieves runtime efficiency between the two uniform-precision baselines.

Compared with ITS, OPT is slightly slower on average, with a runtime increase of only 1.104%. This is expected

	runtime				
	% reduction w.r.t.				
	FP32	**FP16**	**ITS**	**PS**	**RS**
MS	4.804	−0.939	−1.415	8.511	12.955
F1	4.446	−1.973	−0.273	13.994	6.497
RC	3.716	−8.006	−1.109	5.785	6.366
VDC	5.062	−4.293	1.438	4.784	14.186
TTC	13.631	−5.020	−3.147	1.111	6.164
LK	4.390	−0.027	0.653	4.945	15.785
SC	7.258	−0.766	−0.404	6.050	7.680
LC	3.044	−0.731	−0.278	7.765	3.998

TABLE VI: Relative changes of runtime in OPT with respect to baselines (negative value indicates an increment)

because ITS typically performs fewer switches and uses FP16 for a larger number of samples. However, OPT is still faster in benchmarks where the optimized sequence performs only a limited number of switches and executes a large fraction of samples in FP16, such as VDC (see Columns 5–6 in Table IV). For LK, OPT and ITS perform nearly the same number of switches, but surprisingly, OPT has a slightly lower measured runtime than ITS. This is possibly due to runtime fluctuations. In contrast, PS and RS introduce frequent or random switches, which increase switching overhead and degrade runtime across *all* benchmarks. Consequently, OPT reduces runtime by 6.618% and 9.204% on average relative to PS and RS, respectively.

Together, these results show that OPT provides a practical performance-runtime trade-off among all baselines.

***Other Control Performance Metrics**:* We additionally report three other metrics for the optimized precision-switching sequences (OPT): 1) *root mean square error (RMSE)*, 2) *steady-state error (SSE)*, and 3) *quadratic performance index*. We use a discretized version of RMSE, which measures the average deviation of the system output from the desired reference over the simulation horizon [23]. SSE measures the output deviation after the transient response has settled [30]; in our finite-horizon simulations, we compute it up to the simulation limit. The quadratic performance index measures the accumulated state error over time [11]. Lower values of all three metrics indicate better control performance. These metrics are not part of the current optimization objective. We use LQR cost as the primary objective because it is the most principled metric for LQR-based controllers, jointly capturing state deviation and control effort. Nevertheless, our formulation can be naturally extended to optimize or constrain these additional metrics.

We evaluate these metrics only for OPT and the two uniform-precision baselines, FP16 and FP32. This evaluation was motivated by the behavior of LK and MS, where OPT achieves lower LQR cost than FP32. For LK, a similar trend appears for two of the additional metrics: OPT reduces RMSE by 0.146% and the quadratic performance index by 0.121% relative to FP32, while matching FP32 in SSE up to relative differences below 10^{-4}. Interestingly, OPT improves over FP16 in SSE by 20.833%; this is mainly due to strong precision effects in the steady state, particularly in systems like LK having a shorter steady-state phase and more step changes and transients. For MS, the additional metrics do not show the same exception; OPT lies between FP16 and FP32 for all three metrics. Specifically, relative to FP32, OPT increases RMSE by 0.007%, SSE by 0.310%, and the quadratic performance index by 0.014%, while still improving over FP16 by 0.704%, 0.346%, and 0.148%, respectively.

Across all benchmarks, OPT shows improved performance over FP16 and near-FP32 behavior for these additional metrics. On average, OPT reduces RMSE by 0.035%, SSE by 24.626%, and the quadratic performance index by 0.186% relative to FP16. Relative to FP32, OPT shows only small changes: an RMSE increase below $10^{-4}\%$, a 0.039% increase in SSE, and a 0.168% increase in the quadratic performance index. These relative differences are consistent, but smaller than those in the LQR cost, primarily because they measure only output deviations, while the LQR cost simultaneously penalizes the control effort required to achieve the desired reference. Moreover, OPT is optimized for LQR cost rather than for these individual metrics.

RQ4: Multiple Control Tasks Scheduling

In this experiment, we explore how the optimized precision-switching sequences synthesized by our method affect schedulability when multiple control tasks share a single processor.

a) Motivation: Since FP16 is consistently 23.29% faster on average than FP32 in our experiments (Section IV-2), its *worst-case execution time* (WCET) is also expected to be correspondingly smaller. In multi-task settings, using a high precision can render a task set infeasible due to high processor utilization, whereas using a low precision can significantly degrade control performance. Precision switching offers a practical compromise, potentially reducing utilization enough to make an otherwise infeasible task set schedulable, while preserving desired control performances for all tasks.

We assess feasibility using *earliest-deadline-first (EDF)* scheduling, which always schedules the task with the earliest deadline. On a uniprocessor, a task set is EDF-schedulable whenever its total utilization is at most 1 [26].

b) Utilization under Switching Precision: We now formalize the intuition above. Consider n control tasks. Since sampling periods (and thus the number of samples) vary across tasks, we use a common analysis window of $\mathcal{H}$ samples, chosen so that each task executes an integer number of samples within the window (e.g., $\mathcal{H}$ can be the least common multiple of the per-task total samples).

Let h_j denote the sampling period of task j, and let $c_{p,j}$ denote its WCET in precision $p \in \mathcal{P}$. If all n tasks execute entirely in precision p, the processor utilization is: $U_p = \sum_{j=1}^{n} \frac{c_{p,j}}{h_j}$.

With precision switching, task j executes different samples in different precisions according to its synthesized sequence. Let $n_{p,j}$ be the number of samples of task j executed in precision p within $\mathcal{H}$. The corresponding utilization is:

$$U_{sw} \;=\; \sum_{j=1}^{n} \sum_{p=1}^{\mathcal{P}} \frac{c_{p,j} \times n_{p,j}}{\mathcal{H} \times h_j}. \tag{5}$$

Intuitively, $\frac{c_{p,j} \times n_{p,j}}{\mathcal{H} \times h_j}$ is the fraction of processor time consumed by task j when executed in precision p, over the window $\mathcal{H}$. When $U_{sw} \leq 1$, the task set is EDF-schedulable. This condition can hold often, as switching reduces the execution time demands by allowing low precisions in between, compared to a uniform high precision execution.

c) Case Study: We illustrate the potential benefit of precision switching for multi-task scheduling using three benchmark control systems `RC`, `MS`, and `LC` from Section IV-1 that can switch between FP16 and FP32. For each corresponding control task j, we consider the input tuple $(h_j, c_{16,j}, c_{32,j})$ as follows. `RC`: $(20\,\text{ms}, 4.32\,\text{ms}, 5.50\,\text{ms})$, `MS`: $(20\,\text{ms}, 6.77\,\text{ms}, 8.50\,\text{ms})$, `LC`: $(10\,\text{ms}, 2.46\,\text{ms}, 4.25\,\text{ms})$. These assumptions are consistent with the relative per-sample runtime reductions observed for these three systems in Table III. More generally, we treat the WCETs under different precision options as inputs. The total samples of `RC`, `MS`, and `LC` are 150, 150, and 800, respectively, and we choose $\mathcal{H} = \text{lcm}(150, 150, 800) = 1200$.

Executing all three tasks entirely in FP32 yields utilization $U_{32} = \sum_{j=1}^{3} \frac{c_{32,j}}{h_j} = 1.12 > 1$, hence the task set is not EDF-schedulable on a single processor. Executing entirely in FP16 reduces utilization to $U_{16} = \sum_{j=1}^{3} \frac{c_{16,j}}{h_j} = 0.80 \leq 1$, restoring schedulability but at the cost of significantly degraded control performance (Column 5 of Table II).

Using the synthesized precision-switching sequences, we obtain the counts $n_{16,j}$ and $n_{32,j}$ from Column 6 of Table IV and compute:
$U_{sw} = \sum_{j=1}^{3} \frac{c_{16,j} \times n_{16,j} + c_{32,j} \times n_{32,j}}{\mathcal{H} \times h_j} = (\frac{4.32 \times 1032 + 5.50 \times 168}{1200 \times 20} + \frac{6.77 \times 400 + 8.50 \times 800}{1200 \times 20} + \frac{2.46 \times 351 + 4.25 \times 849}{1200 \times 10}) = 0.993 \leq 1$, showing that switching makes this task set EDF-schedulable.

In this setting, the three precision-switching sequences are not only EDF-schedulable on a shared processor, but, for all systems, the resultant EDF-schedule ensures optimal control performance while satisfying the respective settling time requirements. More broadly, a higher fraction of low-precision samples reduces U_{sw} further, potentially allowing additional tasks to be admitted.

This case study highlights a promising direction for *precision-guided multi-controller scheduling*: synthesizing precision-switching sequences that not only balance runtime and control performance per controller, but also improve the schedulability of the overall task set on a shared processor. A natural next step is to extend this approach to multicore systems. In a restricted setting with a fixed mapping from precisions to cores, this can be straightforwardly done by incorporating core-switching overhead. The more general case, with full flexibility over both core assignment and precision selection, remains to be explored.

V. Related Work

Control Scheduling: A substantial body of work has addressed control task scheduling from different perspectives, exploring control characteristics such as control cost, stability, and safety guarantees, analyzing energy efficiency, etc. For example, in the context of skipped or delayed control execution and timing uncertainties, some recent works address aspects such as the system's stability [39], [28], [38] and safety [22] when scheduling a control task. Another recent work [35] proposes energy-based scheduling by switching between multiple sampling periods at runtime.

Moreover, control performance is specifically assessed using metrics such as LQR cost [20], [8], quadratic cost [38], or quadratic performance index [11] in some recent works. Several works have addressed the co-scheduling of multiple control tasks on shared processors using methods such as response-time analysis [11], earliest deadline first (EDF) scheduling [20], [18], or SMT-based formulations [7], [19].

However, none of the scheduling techniques in the literature consider the impact of finite precision on the system's performance requirements, which we introduce for the first time and explicitly address in this work.

Finite-Precision Control and Safety Verification: Several recent works have explored precision considerations in controller implementation and safety verification. For instance, [32] designs robust model-predictive controllers under finite precision, ensuring that deviations in the implemented controller remain within the robustness margin of the original design. In contrast, [13], [27] address the problem from the implementation side by generating sound and efficient implementations—either through heuristic search in a general framework or by formulating and solving a mixed-integer programming problem for neural network controllers. More recent approaches [37], [36] combine finite- and infinite-time safety guarantees with provably safe controller implementations. While these works ensure safety under fixed-point or mixed-precision implementations, they do not consider *when and how* to schedule different precision versions of a controller to balance control performance and computational efficiency.

A recent work [25] shares a similar motivation by exploring precision-aware scheduling of multiple neural network–based real-time tasks. It proposes an idle-aware EDF policy to schedule multiple tasks on CPU–FPGA platforms, allowing each task to choose among different precision levels to maximize accuracy within deadlines. However, unlike our approach, it does not address the performance requirements of controllers.

More broadly, mixed-precision computation has also been studied outside the control context, notably in numerical linear algebra. For example, mixed-precision iterative refinement uses lower precision for costly linear algebra kernels and higher precision for residual computation to improve efficiency while preserving convergence [9], [10]. While conceptually similar in using multiple precisions, they target linear solver convergence rather than real-time closed-loop controller scheduling. In contrast, our work selects one precision per control sample and optimizes switching across samples under control-performance and execution-time constraints.

VI. Conclusion

Precision is an important design choice for controller implementations, and it can significantly influence closed-loop performance. In this paper, we introduce the first closed-loop control scheduling framework with per-sample precision switching. The framework synthesizes when a controller

should switch between floating-point precisions to reduce computational cost while achieving near-optimal control performance and meeting settling-time requirements. We formulate the problem as a multi-objective MIQP and combine sound roundoff-error bounds with exact linearizations to enable efficient solving with state-of-the-art optimizers. Experiments on automotive control benchmarks show that our method efficiently generates FP32/FP16 precision-switching sequences that achieve near-FP32 control performance with total runtimes between the FP16 and FP32 baselines, demonstrating a practical performance-efficiency trade-off. Finally, we show that precision-switching sequences can be relevant for multi-task scheduling on shared processors, where, by improving processor utilization, a feasible schedule can be generated that may not be possible under uniform precision choices. This opens an exciting direction for future precision-aware multi-controller scheduling.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[2] S. Adhikary, I. Koley, S. K. Ghosh, S. Ghosh, et al. Skip to Secure: Securing Cyber-Physical Control Loops with Intentionally Skipped Executions. CPS&IoT Security and Privacy (CPSIoTSec), 2020.

[3] Arm. Cortex-M52 Product Support. https://developer.arm.com/processors/cortex-m52. Accessed: 2026-06-10.

[4] Arm. Cortex-M55 Product Support. https://developer.arm.com/processors/cortex-m55. Accessed: 2026-06-10.

[5] Arm. Cortex-M85 Product Support. https://developer.arm.com/Processors/Cortex-M85. Accessed: 2026-06-10.

[6] K. J. Åström and B. Wittenmark. *Computer-controlled systems*. Prentice-Hall, Inc., 1997.

[7] D. Banerjee, P. S. Duggirala, B. Ghosh, and S. Ghosh. A Formal Approach towards Safe and Stable Schedule Synthesis in Weakly Hard Control Systems. *ACM Transactions on Embedded Computing Systems*, 24(5s):1–26, 2025.

[8] D. Banerjee and S. Ghosh. P2SDS: A Polynomial-Time Pattern-Guided Stable Dynamic Scheduling for Weakly Hard Control Task Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 24(5), 2025.

[9] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak. Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. *The International Journal of High Performance Computing Applications*, 21(4):457–466, 2007.

[10] E. Carson and N. J. Higham. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018.

[11] H. S. Chwa, K. G. Shin, and J. Lee. Closing the Gap between Stability and Schedulability: A New Task Model for Cyber-Physical Systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2018.

[12] E. Darulova, A. Izycheva, F. Nasir, F. Ritter, et al. Daisy - Framework for Analysis and Optimization of Numerical Programs (Tool Paper). Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2018.

[13] E. Darulova, V. Kuncak, R. Majumdar, and I. Saha. Synthesis of Fixed-Point Programs. In *International Conference on Embedded Software (EMSOFT)*, 2013.

[14] A. Das, I. Briggs, G. Gopalakrishnan, S. Krishnamoorthy, and P. Panchekha. Scalable yet Rigorous Floating-Point Error Analysis. High Performance Computing, Networking, Storage and Analysis (SC), 2020.

[15] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine. Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software. International Workshop on Formal Methods for Industrial Critical Systems (FMICS), 2009.

[16] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, et al. The SCIP Optimization Suite 7.0. 2020.

[17] S. Ghosh. Delay-Aware Control for Autonomous Systems. VLSI Design and Embedded Systems (VLSID), 2023.

[18] S. Ghosh, S. Dey, and P. Dasgupta. Performance and Energy Aware Robust Specification of Control Execution Patterns under Dropped Samples. *IET Computers & Digital Techniques (IET-CDT)*, 13(6):493–504, 2019.

[19] S. Ghosh, S. Dey, and P. Dasgupta. Pattern Guided Integrated Scheduling and Routing in Multi-hop Control Networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(2):1–28, 2020.

[20] S. Ghosh, S. Dutta, S. Dey, and P. Dasgupta. A Structured Methodology for Pattern Based Adaptive Scheduling in Embedded Control. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–22, 2017.

[21] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. Accessed: 2024-11-04.

[22] C. Hobbs, B. Ghosh, S. Xu, P. S. Duggirala, and S. Chakraborty. Safety Analysis of Embedded Controllers under Implementation Platform Timing Uncertainties. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 41(11):4016–4027, 2022.

[23] T. O. Hodson. Root Mean Square Error (RMSE) or Mean Absolute Error (MAE): When to Use Them or Not. *Geoscientific Model Development Discussions*, 2022:1–10, 2022.

[24] IBM Corporation. IBM ILOG CPLEX Optimization Studio CPLEX Users Manual, 2024. Accessed: 2024-11-04.

[25] W. Jiang, Z. Song, J. Zhan, Z. He, et al. Optimized Co-scheduling of Mixed-Precision Neural Network Accelerator for Real-Time Multitasking Applications. *Journal of Systems Architecture (JSA)*, 110:101775, 2020.

[26] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[27] D. Lohar, C. Jeangoudoux, A. Volkova, and E. Darulova. Sound Mixed Fixed-Point Quantization of Neural Networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 22(5s):1–26, 2023.

[28] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. Control-System Stability under Consecutive Deadline Misses Constraints. Euromicro Conference on Real-Time Systems (ECRTS), 2020.

[29] M. Moscato, L. Titolo, A. Dutle, and C. A. Munoz. Automatic Estimation of Verified Floating-Point Round-Off Errors via Static Analysis. International Conference on Computer Safety, Reliability, and Security (SAFECOMP), 2017.

[30] K. Ogata. *Modern Control Engineering*. Prentice Hall, 2010.

[31] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty. Multi-Objective Co-optimization of FlexRay-based Distributed Control Systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016.

[32] M. Salamati, R. Salvia, E. Darulova, S. Soudjani, and R. Majumdar. Memory-Efficient Mixed-Precision Implementations for Robust Explicit Model Predictive Control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–19, 2019.

[33] A. Solovyev, M. S. Baranowski, I. Briggs, C. Jacobsen, et al. Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 41(1):1–39, 2018.

[34] M. Sudvarg, A. Clark, and C. Gill. Integrated Real-Time Control and Scheduling for Safety Critical Cyber-Physical Systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2025.

[35] S. A. Tafrishi, X. Dai, Y. Hirata, and A. Burns. Discretization and Stabilization of Energy-Based Controller for Period Switching Control and Flexible Scheduling. American Control Conference (ACC), 2022.

[36] S. Teuber, D. Lohar, and B. Beckert. Of Good Demons and Bad Angels: Guaranteeing Safe Control under Finite Precision. Formal Methods in Computer-Aided Design (FMCAD), 2025.

[37] H. Thevendhriya, S. Ghosh, and D. Lohar. Toward Precision-Aware Safe Neural-Controlled Cyber–Physical Systems. *IEEE Embedded Systems Letters (ESL)*, 16(4):397–400, 2024.

[38] N. Vreman, A. Cervin, and M. Maggio. Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses. Euromicro Conference on Real-Time Systems (ECRTS), 2021.

[39] N. Vreman, C. Mandrioli, and A. Cervin. Deadline-Miss-Adaptive Controller Implementation for Real-Time Control Systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2022.

[40] G. v. Zengen, J. Yu, and L. C. Wolf. Adaptive Real-Time Scheduling for Cooperative Cyber-Physical Systems. Industrial Cyber-Physical Systems (ICPS), 2020.