This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674

A Hybrid Approach to Refine WCRT Bounds for DAG Scheduling Using Anomaly Classification

Nan Chen, Xiaotian Dai, Alan Burns, Iain Bate
Department of Computer Science, University of York, UK

Abstract—Motivated by the performance demands and stringent timing requirements of safety-critical systems like avionics and autonomous vehicles, research has focused on providing timing guarantees for the scheduling of Directed Acyclic Graph (DAG) tasks in multicore systems. The structural complexity and timing anomalies make this problem challenging. Existing methods bound the Worst-Case Response Time (WCRT) of tasks through static analysis, but these bounds are complicated, difficult to validate, and often remain pessimistic for many scheduling scenarios. Runtime intervention can be effective in eliminating timing anomalies and providing timing guarantees; however, it is ineffective for anomaly-free scheduling scenarios, leads to non-work-conserving schedules, and incurs additional overhead. This paper proposes a hybrid approach to identify timing anomalies in DAG scheduling scenarios within a system, providing tighter WCRT solutions. The static analysis first offers a sufficient anomaly test to directly identify some anomaly-free DAG scheduling scenarios. Leveraging a wide range of scheduling data collected from the running system or its simulator, we then apply a machine learning approach to train a binary classification model, achieving an accuracy of 99.5%. Identifying the anomaly status enables the application of more precise WCRT bounds for different scheduling scenarios, leading to improved system performance. Specifically, we shorten the WCRT bounds for anomaly-free DAG scheduling by an average of up to 21.58%, with a maximum reduction of up to 55.47% compared to the state-of-the-art method.

Keywords: DAG scheduling, timing anomaly, machine learning, real-time systems

1 Introduction

The growing complexity and strict timing demands in automotive and avionic real-time systems exemplified by Tesla's Full Self-Driving (FSD) computer[1], highlights the need for advanced planning and navigation, with robust real-time capabilities. These systems must operate flawlessly and their tasks must meet their deadlines to ensure the safety and efficiency of autonomous vehicles in precision-critical environments with minimal human oversight. These requirements drive the need to effectively manage complex task models with dependencies in multicore systems, among which many computational workloads, such as TensorFlow[2] and data processing pipelines in Apache Spark [1], can be effectively represented as Directed Acyclic Graphs (DAGs), where nodes often referred to as individual processing units, and edges indicate the dependencies between these units. The branching structure within a DAG illustrates the inherent parallelism within such workloads.

The complexity of DAG structures and multicore interactions leads to timing anomalies inherent in global work-conserving DAG scheduling with a limited preemption scheme (see Section 3), making timing guarantees challenging. In a scheduling scenario (i.e., a given DAG task scheduled on a specific number of cores), timing anomalies occur when nodes execute for a shorter time than expected at runtime, but the shorter local execution times paradoxically lead to a longer overall makespan due to changes in execution order [2].

Extensive research has been conducted to provide timing guarantees for the scheduling of DAG tasks. A significant proportion [3]–[6] develop Worst-Case Response Time (WCRT) bounds through static analysis. These bounds assume that the Worst-Case Execution Time (WCET) of each node within the DAG can be upper bounded by static analysis [7]. WCRT bounds are determined by considering the worst possible interference and blocking between nodes in a scheduling scenario and calculating the makespan without being affected by anomalies. However, these static generic bounds face bottlenecks: simpler bounds [3], [4] incur high pessimism, limiting system performance; sophisticated bounds [6] are difficult to validate and have known flaws [8], with pessimism still existing in certain scheduling scenarios.

Instead of providing WCRT bounds, studies have introduced runtime approaches to eliminate timing anomalies in DAG scheduling [9]–[11]. One such approach involves monitoring the actual runtime execution of each node within a DAG and idling the core to preserve the pre-established execution order [10]. While this method provides timing predictability without complicated analysis, it renders the system non-work-conserving, as ready nodes may remain idle even when a core is available. Despite these efforts, such a solution is ineffective, especially for those scheduling scenarios without timing anomalies. Additionally, online monitoring introduces overhead, potentially impacting system performance. Therefore, providing tailored WCRT solutions for various scheduling scenarios is demanded to reduce pessimism while guaranteeing overall safety, but a more comprehensive understanding of the system is required.

Contributions: This paper provides timing guarantees for DAG scheduling on multicore systems by introducing a hybrid approach to identify the anomaly status of different DAG scheduling scenarios and provide tighter WCRT bounds. Our contributions can be listed as follows:

- A comprehensive literature review is conducted, focusing on

1. https://www.tesla.com/engb/support/autopilo
2. https://www.tensorflow.org/

the development of timing guarantees in DAG scheduling.

- We develop a novel sufficient anomaly test based on a simulation algorithm and static analysis to identify scheduling scenarios without timing anomalies.
- Leveraging on the scheduling data collected from the running system or its simulator, utilising the identified anomaly-free scheduling scenarios based on the anomaly test and observed anomaly cases, we apply a machine learning approach to train a binary classification model that can classify the anomaly status of scheduling scenarios with an accuracy of 99.5%.
- Extensive evaluation results show that the proposed approach allows the application of different WCRT bounds and shortens WCRT bounds for anomaly-free scheduling by an average of up to 21.58%, with a maximum reduction of up to 55.47% compared with the state-of-the-art.

Organisation: This paper is structured as follows: Section 2 reviews the literature. Section 3 provides detailed definitions of and nomenclature for the system and task models. Section 4 introduces the technical details of the proposed hybrid analysis approach. The evaluation results are demonstrated in Section 5, and finally, Section 6 concludes the paper.

2 Related Work

The inherent complexity of DAG structures makes it challenging to provide timing guarantees under multicore scheduling. In the meantime, partitioned scheduling offers a more deterministic allocation of DAG nodes and helps mitigate uncertainty in execution. Fonseca et al. [12] focus on the partitioned scheduling of sporadic DAG tasks, developing WCRT bounds under static task allocation to provide schedulability guarantees. Several subsequent works extend this line of research, exploring slight variations in task and scheduling models [13], [14]. However, while partitioned scheduling enhances predictability, it also restricts flexibility and may lead to reduced system utilization. In contrast, global or federated scheduling introduces more complexity but can potentially achieve higher system utilisation.

Many studies have examined global or federated work-conserving scheduling, where ready nodes are assigned to cores as they become available. Bonifaci et al. [15] compare Earliest Deadline First (EDF) and Deadline Monotonic (DM) scheduling for DAG tasks in multicore systems under global work-conserving conditions. They derive speedup bounds for both strategies. Building upon this, Baruah improves the speedup bound for global EDF scheduling in [16]. Similarly, Qamhieh et al. [17] propose a *stretching* algorithm that preserves task dependencies through timing constraints, further improving the speedup bound under global EDF.

Baruah [18] then turns to federated scheduling, extending previous work to arbitrary-deadline DAG tasks. By dedicating processors to high-density tasks and partitioning the rest, the proposed algorithm achieves a tight speedup bound. In a follow-up, Baruah [19] extends the federated model to support conditional sporadic DAG tasks, where each task may execute one of several possible DAGs at runtime. Subsequently, Baruah et al. [20] revisit global EDF scheduling for conditional sporadic DAGs. They develop a speedup bound that accounts for execution path uncertainty, allowing more dynamic utilization of shared multicore resources. Wang et al. [21] further advance global EDF scheduling for DAG tasks with arbitrary deadlines. Their proposed analysis techniques yield improved capacity augmentation bounds in single-task scenarios.

Beyond speedup and augmentation bounds, many studies have derived WCRT bounds for DAG tasks under global scheduling, typically assuming simple preemptive scheduling policies. Melani et al. [3] assume fixed-priority scheduling and develop a response-time analysis framework for conditional DAG tasks, providing tighter and more general WCRT bounds, which also apply to conventional DAGs. Fonseca et al. [22], [23] further reduce interference through structural analysis of DAGs, resulting in improved bounds. More recently, He et al. [24]–[26] focus on single DAG tasks, progressively refining WCRT bounds across multiple papers by precisely capturing DAG paths and minimizing interference.

Serrano et al. [4] investigate WCRT bounds under limited preemption, where a node, once started, cannot be preempted. This assumption better reflects practical systems, since preemption at the level of individual nodes often incurs costly context switches. Similarly, Nasri et al. [27] address WCRT analysis for limited-preemptive DAG tasks but diverge from conventional methods by adopting a richer task model, including release jitter, execution-time uncertainty, and inter-task dependencies.

Most of the aforementioned works apply task-level priority schemes. He et al. [5] introduce intra-task priority assignment, showing that prioritizing individual nodes can reduce interference and significantly tighten WCRT bounds with preemptive scheduling. This approach is further refined in [28] through a more effective priority assignment strategy. Zhao et al. [6] extend the intra-task ordering concept to limited-preemption scheduling by introducing a provider-consumer model that guides priority assignment and WCRT analysis. Their follow-up work [29] adapts this approach to federated scheduling, supporting multiple DAG tasks. However, Chen et al. [8] identify a flaw in the WCRT analysis presented in [6], which remains unresolved in [29]. In response, Chen et al. propose a new analysis technique for both single- and multi-DAG task systems. Their method accurately computes the worst-case finish time of each node, setting a new state-of-the-art bound under global work-conserving with limited preemption. Finally, He et al. [30] present the first application of intra-task fixed-priority assignment to conditional DAG tasks under preemptive scheduling. Instead of a conventional analysis approach, Chen et al. [10] provide WCRT bounds by simulating DAG execution under global scheduling with limited preemption. Their dynamic scheduling strategy maintains node execution order to prevent timing anomalies and ensure timing guarantees. Sun et al. [31] formulate WCRT computation as an SMT optimisation problem to determine the exact WCRT bound.

DAG scheduling has also been extended to accommodate more complex task models. Bi et al. [32] incorporate mutually exclusive nodes into the prioritized DAG model and derive rigorous WCRT bounds. In contrast, Liang et al. [33] proactively assign mutually exclusive groups to DAG nodes, transforming scheduling behaviour and enabling WCRT analysis via a critical mutual path without explicitly modeling interference. Finally, the survey by Verucchi et al. [34] provides a comprehensive review of recent advances in DAG task scheduling for real-time systems. It compares global and partitioned approaches, highlighting their theoretical foundations, practical effectiveness, and industrial applicability.

Many challenges in the field of real-time systems are effectively addressed by more than single approach. For instance, Jones et al. [35] propose a hybrid approach for real-time sequencing and scheduling by integrating neural networks, simulations, genetic

TABLE 1: Table of notations.

Notation	Description
Γ	Set of sporadic tasks
τ_i	A given task with index i
$J_{i,1}$	A job instance of τ_i with index 1
$\mathcal{G}_i = (V_i, E_i)$	Graph defining the set of activities forming task τ_i
V_i	Set of nodes in τ_i
E_i	Set of directed edges connecting nodes in τ_i
v_j, v_x	Node with index j, x in DAG task τ_i
C_{τ_i}	Total WCET of nodes within the DAG
C_{v_j}, C'_{v_j}	WCET, varying execution time of node v_j
P_{v_j}	Priority of node v_j
$pre(v_j), suc(v_j)$	Set of predecessors, successors of node v_j
$anc(v_j), des(v_j)$	Set of ancestors, descendants of node v_j
v_{src}, v_{sink}	Source node, sink node
f'_{v_j}, s'_{v_j}	Varying finish time, start time of node v_j
$f^{pre}_{v_j}$	Time when node v_j is eligible to execute after all its predecessors finish
Λ, $\lvert\Lambda\rvert$, λ_m	Set of cores, the total number of cores, core with index m in the system
$Con(v_j)$	Set of concurrent nodes of node v_j
$f^{\lvert\Lambda\rvert}_{v_j}$, $s^{\lvert\Lambda\rvert}_{v_j}$	Finish time, start time of node v_j in the long-makespan scenario

algorithms, and a trace-driven knowledge acquisition technique. Lampka et al. [36] tackle the complexity of analysing embedded real-time systems by combining timed automata for state-based timing accuracy with analytic methods for system simplicity. Reghenzani et al. [37] tackle the problem of estimating WCET in safety-critical embedded systems using a probabilistic approach. They leverage Measurement-Based Probabilistic Timing Analysis (MBPTA) and Extreme Value Theory (EVT) to model and predict extreme execution times.

Overall, current research provides timing guarantees for the scheduling of DAG tasks through purely static analysis. This research primarily relies on offline assumptions of system information, inevitably incorporating considerable pessimism. Although the WCRT bounds are gradually improving through more detailed static analysis, this progress comes at the cost of increasing complexity and presents challenges in verifying their correctness. Monitoring and preserving the execution order of nodes within DAG tasks can provide timing predictability for the scheduling. However, it incurs runtime overheads and makes the system non-work-conserving. To the best of our knowledge, this is the first paper that incorporates both static analysis and machine-learning approaches to provide timing guarantees for DAG scheduling.

3 System Model

3.1 Task model

We focus on a system with a set of sporadic tasks Γ, in which τ_i represents a given task with index i. Each τ_i generates a potentially unbounded sequence of job instances $J_{i,1}, J_{i,2}, ..., J_{i,z}$. As each job is only allowed to be executed at a time, hence we refer to a task and its job as equivalent entities in terms of execution within the system. Each task in the system is a DAG task and is defined by $\{C_{\tau_i}, T_i, D_i, \mathcal{G}_i = (V_i, E_i)\}$. Here, C_{τ_i} is the total WCET of nodes within the DAG, T_i denotes its period (or the minimum inter-arrival time between jobs), D_i provides a constrained relative deadline, i.e., $D_i \leq T_i$, and $\mathcal{G}_i$ is a graph that defines the set of activities forming the task. The graph is further detailed as $\mathcal{G}_i = (V_i, E_i)$, where V_i denotes the set of nodes, and $E_i \subseteq (V_i \times V_i)$ represents the set of directed edges connecting any two nodes. A node in τ_i is denoted as $v_{i,j} \in V_i$, where the index j specifies the node index, and index i indicates that it belongs to τ_i. For simplicity, the subscript of the DAG task (i.e., i for τ_i) is omitted when discussing a single DAG task.

The WCET of a node v_j is denoted as C_{v_j}. In reality, nodes are likely to execute less than their WCET; we denote their varying execution times in different releases as C'_{v_j}. Each node v_j is assigned an individual priority, denoted as P_{v_j} (the larger the number, the higher the priority). If two nodes, v_j and v_x, are connected by a directed edge (i.e., $(v_j, v_x) \in E$), v_x can only start executing after v_j has finished executing. Thus, v_j is a predecessor of v_x, and v_x is a successor of v_j.

The predecessors and successors of node v_j can be formally defined as $pre(v_j) = \{v_x | v_x \in V_i \wedge (v_x, v_j) \in E\}$ and $suc(v_j) = \{v_x | v_x \in V_i \wedge (v_j, v_x) \in E\}$, respectively. Nodes that are either direct or transitive predecessors and successors of a node v_j are termed its ancestors $anc(v_j)$ and descendants $des(v_j)$, respectively.

A node v_j with $pre(v_j) = \varnothing$ or $suc(v_j) = \varnothing$ is referred to as the source node v_{src} or sink node v_{sink}, respectively. As with most existing work [5], [6], [28], we assume a DAG task always starts with one source and ends with one sink node; otherwise, a dummy node without utilisation is added to the beginning or the end of the DAG.

3.2 System and scheduling model

We focus on a homogeneous multicore system. From the task level, the scheduler allocates a set of cores Λ to schedule a DAG, where $|\Lambda|$ defines the total number of cores and λ_m determines the core with an index m. These cores are not utilised until the allocated DAG exits the system. For nodes within a DAG, we apply a global fixed-priority scheduling method with the limited preemption scheme (nodes are non-preemptive once they start executing) in a work-conserving manner [4]. For all nodes ready to execute, the scheduler follows the highest-priority-first execution order and keeps dispatching nodes when a core idles. We do not presume any priority assignment for nodes.

Relevant notations are summarised in Table 1. We further introduce the additional definitions and provide an intuitive example of scheduling a DAG in Example 1 to aid the explanation throughout the paper.

Definition 1. ***Concurrent nodes [8]:*** *For a pair of nodes $v_j, v_x \in V_i$, if v_x is neither the ancestors of v_j (i.e. $v_x \notin anc(v_j)$) nor the descendants of v_j (i.e. $v_x \notin des(v_j)$), it is referred to as a concurrent node of v_j. The set of concurrent nodes of v_j is denoted as $Con(v_j)$ and is expressed as Equation 1.*

$$Con(v_j) = \{v_x | v_x \in V_i \wedge v_x \notin anc(v_j) \wedge v_x \notin des(v_j) \wedge v_x \neq v_j\} \quad (1)$$

Definition 2. ***Delay:*** *A node is considered delayed when it is eligible to execute but is prevented from doing so due to blocking by concurrent low-priority nodes, interference from concurrent high-priority nodes, or both. Direct delay refers to the immediate delay experienced by the node, while indirect delay is incurred transitively through its predecessors or ancestors.*

Definition 3. ***Makespan:*** *For a given DAG task with nodes executing with varying execution times up to their WCETs, the finish time of the sink node denotes the makespan of the task in*

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674

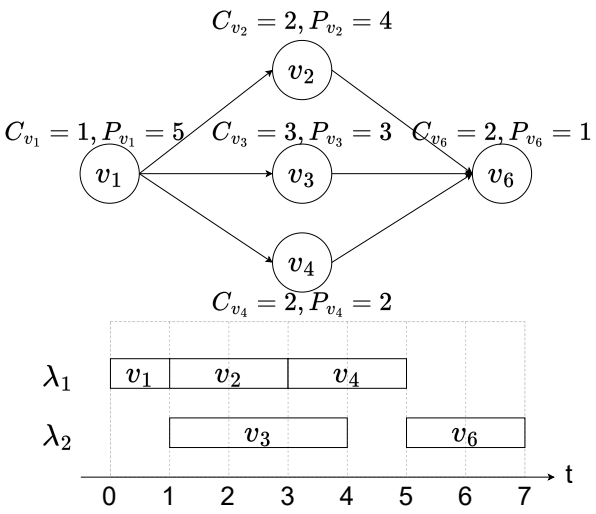


Fig. 1: Scheduling of a DAG task.

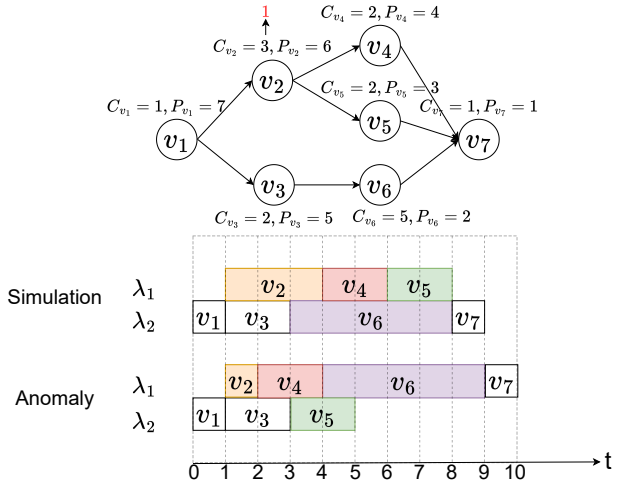


Fig. 2: An example of timing anomaly [10].

any given scenario. The WCRT of a DAG determines its worst-case makespan in any possible scenario.

Definition 4. ***Scheduling scenario****: In this paper, a scheduling scenario refers to a DAG task scheduled on a given number of cores. The execution times of nodes in a scheduling scenario can vary across different releases. The varying start and finish times for v_j are denoted as s'_{v_j} and f'_{v_j}, respectively. The time when a node is eligible to execute after all predecessors have finished is denoted as $f^{pre}_{v_j}$.*

Definition 5. ***Long-makespan scenario****: The scheduling scenario where all nodes execute up to their WCETs. In a long-makespan scenario, the start and finish time of a given node v_j are denoted as $s^{|\Lambda|}_{v_j}$ and $f^{|\Lambda|}_{v_j}$ respectively.*

Example 1. *Figure 1 shows the scheduling of a DAG task on a dual-core system in our scheduling context; v_1 is the source node, and v_6 is the sink node. The descendants of v_1 are $des(v_1) = \{v_2, v_3, v_4, v_6\}$, and the ancestors of v_6 are $anc(v_6) = \{v_1, v_2, v_3, v_4\}$. Taking v_3 as an example, its WCET is $C_{v_3} = 3$, its priority is $P_{v_3} = 3$, and its concurrent nodes are $Con(v_3) = \{v_2, v_4\}$. The predecessors and successors of v_3 are $pre(v_3) = \{v_1\}$ and $suc(v_3) = \{v_6\}$.*

Assuming each node executes up to its WCET (i.e. long-makespan scenario), as shown in the figure, v_1 initially executes on core λ_1 until $t = 1$. At this time, nodes $\{v_2, v_3, v_4\}$ become eligible to execute. Since $\{v_2, v_3\}$ have the highest priorities, they are scheduled first; v_4 begins at $t = 3$ on λ_1 in a work-conserving manner. Then v_6 starts at $t = 5$ and ends at $t = 7$ resulting in the makespan of the DAG equal to 7 in this scenario.

Definition 6. ***Timing anomaly****: The timing anomaly (or anomaly) refers to a DAG task in a scheduling scenario that has a longer makespan than its long-makespan scenario. An example is illustrated in Example 2.*

Example 2. *Figure 2 presents a DAG task scheduling on a dual-core system and its two abstracted execution Gantt charts (Simulation and Anomaly). As shown in the Simulation chart, all nodes follow the scheduling rule that we defined in Section 3 and each node executes up to its WCET which results in a simulation makespan for the DAG of 9 units. However, in the Anomaly scenario, v_2 executes for only one unit of time, from $t = 1$ to $t = 2$, which initially appears to be an improvement. However, due to limited preemption, this alters the execution order of v_4, v_5, and v_6, resulting in a longer makespan of 10 units.*

4 A Hybrid Analysis Approach

In this section, we introduce a hybrid analysis approach to identify the anomaly status of different DAG scheduling scenarios. First, we will formulate the problem we are targeting. Next, we will introduce an anomaly test built through the static analysis approach, followed by proposing a binary classification model.

4.1 Problem formulation

As illustrated by Definition 6 and Example 2, the timing anomaly in a scheduling scenario occurs when the actual makespan turns out to be longer than its long-makespan scenario. Conversely, if we can ensure that a scheduling scenario does not exceed its long-makespan scenario, which is considered anomaly-free in this work, we can upper bound the WCRT of the scheduling scenario with the simulated long-makespan scenario. This method likely yields a tighter bound than conventional static analysis because the simulated bound is an exact bound [10]. For scheduling scenarios with the anomaly, traditional bounds can still ensure safety. *Therefore, our goal is to develop a hybrid analysis approach to reliably identify the anomaly status of different scheduling scenarios.*

4.2 Anomaly test with static analysis

In this subsection, we describe the first part of our hybrid approach, which utilises a static analysis method to build an anomaly test based on the simulated long-makespan scenario. The anomaly test can identify whether a DAG task in a given scheduling scenario can overrun its long-makespan scenario, thereby determining its anomaly status.

The prerequisite step of constructing the anomaly test is to develop an algorithm presented in Algorithm 1 to simulate the long-makespan scenario of a given DAG task on a fixed number of cores. This algorithm determines the start and finish times of

each node, as well as the overall makespan of the DAG. We do not claim this is an optimal algorithm for simulating the long-makespan scenario.

Algorithm 1: Long-makespan scenario simulation

```
1  Λ ← {λ_1, λ_2, ..., λ_|Λ|}, Q ← {v_1, v_2, ..., v_n},
     FinishTimes ← {}, StartTimes ← {}
2  while Q ≠ ∅ do
3      Sort Λ by load(λ) in ascending order;
4      R ← {v_j ∈ Q : v_j.isReady()} ;
5      Sort R by P_{v_j} in descending order;
6      allocated ← FALSE;
7      foreach v_j ∈ R do
8          if load(Λ[1]) ≥ f^pre_{v_j} then
9              StartTimes[v_j] ← load(Λ[1]);
10             Add v_j to Λ[1];
11             load(Λ[1]) ← load(Λ[1]) + C_{v_j};
12             FinishTimes[v_j] ← load(Λ[1]);
13             Remove v_j from Q;
14             allocated ← TRUE;
15             break;
16         end
17     end
18     if not allocated then
19         v_min ← node in R with minimum f^pre_{v_j};
20         StartTimes[v_min] ← f^pre_{v_min};
21         Add v_min to Λ[1];
22         load(Λ[1]) ← f^pre_{v_min} + C_{v_min};
23         FinishTimes[v_min] ← load(Λ[1]);
24         Remove v_min from Q;
25     end
26 end
27 makespan ← max(values of FinishTimes);
28 return FinishTimes, StartTimes, makespan;
```

The algorithm begins with an initialisation phase (line 1), setting up the processing cores Λ, the node queue Q with nodes from the DAG task τ_i, and arrays $StartTimes$ and $FinishTimes$ to store the start and finish times of each node.

The key of the algorithm is a scheduling loop (line 2), which continues until all nodes are scheduled. Each iteration sorts the cores by workload ($load(\lambda)$ in line 3) to prioritise the least loaded core ($load(\Lambda[1])$) for the next node allocation, ensuring a work-conserving approach. Here, workload refers to the amount of execution time already assigned to each core. The algorithm then identifies the subset of nodes R that are ready for execution based on their dependencies (line 4), as a node is ready if all its predecessors have been scheduled.

Within this loop, the algorithm sorts the ready nodes R by priority (line 5), ensuring that higher-priority nodes are scheduled first. It attempts to allocate a node according to the sorted order from R to the least loaded core (lines 7-17). A node is assigned to a core if the core's workload is sufficient to start the node immediately, based on the node's maximum predecessor finish time $f^{pre}_{v_j}$ (line 8). If the condition is met, both the node's times and the core's workload are updated. The node is removed from Q and this is deemed a successful allocation(lines 9-13).

If no nodes in the ready queue can be allocated to the least loaded core due to unresolved dependencies, the node with the earliest $f^{pre}_{v_j}$ is assigned to the core (lines 18-25). This ensures that the node with the earliest eligible execution time is scheduled.

Finally, the algorithm calculates the makespan as the maximum value in $FinishTimes$ (line 27) and returns the start and finish times for each node along with the makespan, providing a complete snapshot of the scheduling process.

The overall time complexity of Algorithm 1 is governed by its scheduling loop. Initially, the algorithm prepares n nodes from the DAG task τ_i in the queue Q. Each iteration of the while-loop removes exactly one node from Q, either through successful allocation (Lines 7–13) or fallback scheduling (Lines 18–24). As no node is revisited, the loop executes exactly n times. Within each iteration, sorting the cores Λ by workload requires $O(|\Lambda|\log|\Lambda|)$ time; since $|\Lambda|$ is typically a small constant, this is negligible in asymptotic analysis. Identifying the ready set $R \subseteq Q$ involves checking dependencies for up to n nodes, requiring $O(n)$ time. Sorting R by priority incurs $O(n \log n)$ time in the worst case. The subsequent allocation loop and fallback selection each take at most $O(n)$ time. Thus, each iteration takes $O(n \log n)$ time, leading to an overall worst-case time complexity of $O(n^2 \log n)$. In practice, due to a sparsely populated ready set and a small number of cores, the observed complexity often approaches $O(n^2)$.

Following Algorithm 1, we propose the following lemma to validate the fixed timing metrics of the long-makespan scenario which allows direct referencing for the following analysis.

Lemma 1. *For a DAG task scheduling on a given number of cores $|\Lambda|$ in the system described in Section 3, assuming each node (say v_j) executes up to its WCET, we will have the following fixed timing metrics in the long-makespan scenario: the start ($s^{|\Lambda|}_{v_j}$) and finish ($f^{|\Lambda|}_{v_j}$) times of each node v_j, as well as the makespan of the DAG.*

Proof. Consider the following invariant conditions within the scheduling: 1) The number of processing cores is fixed, limiting the maximum concurrent node execution and setting an upper bound on throughput; 2) The DAG structure and node priorities remain unchanged, ensuring a consistent sequence for nodes becoming ready; and 3) The scheduling algorithm is work-conserving and non-preemptive, meaning: (a) all cores are utilised as long as there are ready nodes, preventing idling, and (b) nodes with constant execution times run to completion without interruption, maintaining a consistent timeline. Therefore, under these conditions, the lemma holds. □

The development of Algorithm 1 allows us to get the $s^{|\Lambda|}_{v_j}$ and $f^{|\Lambda|}_{v_j}$ of each node v_j. Based on the metrics, an anomaly test is built through static analysis to analyse whether each node can finish later than its forecasted finish time ($f^{|\Lambda|}_{v_j}$) in any circumstances. If any node is found that can potentially execute beyond its forecasted finish time, we identify that the entire DAG is susceptible to timing anomalies under such a scheduling scenario. Prior to beginning the analysis, we stipulate the following assumption:

Assumption 1. *When we analyse a given node, v_j, in the long-makespan scenario, it is assumed that all nodes, denoted as v_x, which start before v_j ($s^{|\Lambda|}_{v_x} < s^{|\Lambda|}_{v_j}$), have already passed their anomaly tests and will not finish later than their predicted finish times, $f^{|\Lambda|}_{v_x}$. This condition is consistently achievable by analysing the nodes in the order of their starting times.*

According to the Example 2, v_2 finishes earlier, which advances the starting times of v_4 and v_5 but delays v_6, causing it to finish later than its $f^{|\Lambda|}_{v_6}$. Based on this observation, we define the following lemma with a more detailed analysis.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674

Lemma 2. *For a given node v_j, it may finish later than its $f_{v_j}^{|\Lambda|}$. The only reason for this delay is the early completion of some nodes within the DAG could potentially bring forward the eligible starting times of other nodes, while delaying v_j due to limited core resources and non-preemptive constraints, ultimately causing v_j to finish later.*

Proof. For a given node v_j, its finish time depends on three factors: 1) the execution time of its $anc(v_j)$; 2) the direct or indirect interference or blocking; and 3) the execution time of v_j. The shortening of execution times for $anc(v_j)$ and v_j will only directly shorten f'_{v_j}. Therefore, the only reason for an increase in f'_{v_j} is due to additional direct or indirect interference or blocking of v_j. This occurs because some nodes, say v_x, start earlier than their $s_{v_x}^{|\Lambda|}$, which consequently delays v_j. □

According to Lemma 2, our goal is to determine which type of node can potentially start earlier and delay a given node $v_j \in V_i$, leading to $f'_{v_j} > f_{v_j}^{|\Lambda|}$. First, we bound the number of nodes that can introduce either interference or blocking, which could delay v_j. In the following analysis, Equations 2,3 and 4 are derived from the state-of-the-art response time analysis of DAG scheduling [8]. We build upon this analysis to ensure accuracy and provide more in-depth insights based on the remaining equations, where we present our novel contributions.

According to [8], only the concurrent nodes $Con(v_j)$ can impose interference or blocking delays on v_j. More precisely, according to Lemmas 4 and 5, and Theorem 1 in [8], considering the dependency and priority constraints, the amount of possible delay can be narrowed down to the nodes contained within $\mathcal{IB}_{v_j}^{potential}$ as shown in Equation 2:

$$\mathcal{IB}_{v_j}^{potential} = Con(v_j) \setminus \mathcal{IB}_{v_j}^{remove} \tag{2}$$

As shown in Equation 3, $\mathcal{IB}_{v_j}^{remove}$ is introduced to identify nodes that cannot impose interference or blocking delays on v_j. This set iterates over all nodes in the DAG ($v_x \in V_i$) and includes a given v_x and $des(v_x)$ if either of the following two conditions are met: 1) the priority of v_x is lower, i.e., $P_{v_x} < P_{v_j}$ and $v_x \in \eta(v_j)$, or 2) $P_{v_x} < P_{v_j}$ and the intersection of ancestors of v_x and $\eta(v_j)$ is non-empty, denoted by $anc(v_x) \cap \eta(v_j) \neq \emptyset$.

$$\mathcal{IB}_{v_j}^{remove} = \bigcup_{v_x \in V_i} \begin{cases} v_x \cup des(v_x), & if\ P_{v_x} < P_{v_j} and \\ & (v_x \in \eta(v_j) \\ & or\ anc(v_x) \cap \eta(v_j) \neq \emptyset) \\ \emptyset, & \text{otherwise.} \end{cases} \tag{3}$$

To understand Equation 3, we break down each notation included. $\eta(v_j)$ is introduced in Equation 4 which includes nodes $v_x \in V_i$ while satisfying $pre(v_j)$ is a subset of $pre(v_x)$ which means v_x is always eligible to execute at the same time or later than v_j (i.e. $f_{v_x}^{pre} \geq f_{v_j}^{pre}$).

$$\eta(v_j) = \left\{ v_x | v_x \in V_i \wedge pre(v_j) \subseteq pre(v_x) \wedge v_j \neq v_x \right\} \tag{4}$$

Revising the conditions specified in Equation 3, the first condition asserts that v_x is a lower-priority node that is eligible to execute with or later than v_j. The second condition states that while v_x is a lower-priority node, its ancestor is eligible to execute with or later than v_j, which implies that v_x itself is always eligible to execute later than v_j. Therefore, both rules indicate all nodes in $\mathcal{IB}_{v_j}^{remove}$ cannot delay v_j, because they are eligible to execute later or at the same time with v_j and the scheduler will always schedule v_j first according to the highest-priority-first scheduling rule denoted in Section 3. An intuitive example to illustrate $\mathcal{IB}_{v_j}^{remove}$ is provided in Example 3.

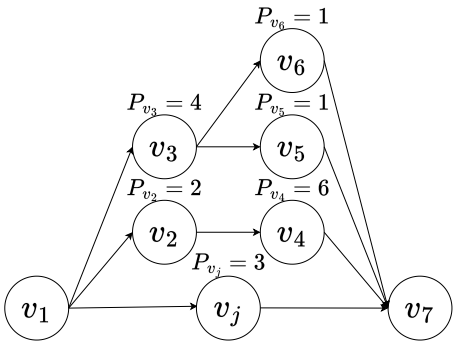


Fig. 3: An example of bounding $\mathcal{IB}_{v_j}^{remove}$.

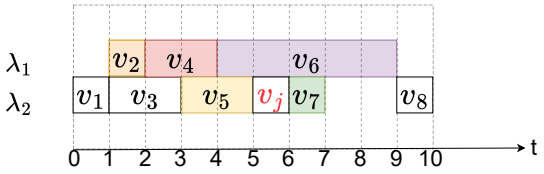


Fig. 4: An example of bounding $\mathcal{IB}_{v_j}^{final}$.

Example 3. *As shown in Figure 3, for a given node v_j in a DAG task, its concurrent nodes are $Con(v_j) = \{v_2, v_3, v_4, v_5, v_6\}$. Then $\eta(v_j) = \{v_2, v_3\}$, which are eligible to execute at the same time as v_j since they share the same predecessor. For v_2, because $P_{v_2} < P_{v_j}$, the scheduler will prioritise v_j for scheduling. Consequently, although $P_{v_4} > P_{v_j}$, $v_4 \in des(v_2)$ and cannot interfere with v_j under any circumstances. Therefore, $\{v_2, v_4\} \in \mathcal{IB}_{v_j}^{remove}$ (condition 1 in Equation 3). For $\{v_6, v_5\}$, because their ancestor $v_3 \in \eta(v_j)$, they are eligible to execute after v_j. Since their priorities $P_{v_5}, P_{v_6} < P_{v_j}$, $\{v_5, v_6\} \in \mathcal{IB}_{v_j}^{remove}$ (condition 2 in Equation 3). Overall, $\mathcal{IB}_{v_j}^{remove} = \{v_2, v_4, v_5, v_6\}$ in this example.*

According to Lemma 2, the only possible way for a given node v_j to execute longer than its $f_{v_j}^{|\Lambda|}$ is if some nodes are brought forward to execute before v_j. Therefore, we further exclude those nodes that finish before or at the same time as the starting time of v_j ($s_{v_j}^{|\Lambda|}$), as shown in Equation 5. Moreover, according to Assumption 1, these nodes cannot finish later than their $f_{v_x}^{|\Lambda|}$, hence they cannot further delay the start time of v_j. An example is illustrated in Example 4.

$$\mathcal{IB}_{v_j}^{final} = \mathcal{IB}_{v_j}^{potential} \setminus \{v_x | v_x \in \mathcal{IB}_{v_j}^{potential} \wedge f_{v_x}^{|\Lambda|} \leq s_{v_j}^{|\Lambda|}\} \tag{5}$$

Example 4. *Figure 4 shows a scheduling example of a given DAG task by Algorithm 1 on a dual-core system, with all nodes executed up to their WCETs. Assume $\mathcal{IB}_{v_j}^{potential} = \{v_2, v_4, v_5, v_6, v_7\}$ in this example. As shown in the figure, $\{v_2, v_4, v_5\}$ finishes before the time point $t = 5$ which is the start point of v_j. Assuming they*

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674

are deemed anomaly-free according to Assumption 1. Therefore, they cannot further delay v_j themselves. However, if they execute shorter than their WCETs, there is a potential risk of bringing the execution of v_7 forwards and delaying v_j. Therefore, $\mathcal{IB}_{v_j}^{final} = \{v_6, v_7\}$.

Moreover, for nodes in $\mathcal{IB}_{v_j}^{final}$ to start earlier and delay v_j, they should take up all cores at the time point $s_{v_j}^{|\Lambda|}$ (when v_j starts in the simulation) and force v_j to start later. Therefore, Lemma 3 is addressed to formalise this condition.

Lemma 3. *For nodes in $\mathcal{IB}_{v_j}^{final}$ to impose either interference or blocking delay to v_j, there should be at least $|\Lambda|$ number of nodes that can execute in parallel.*

Proof. For v_j to finish later than its $f_{v_j}^{|\Lambda|}$, it must start later than its $s_{v_j}^{|\Lambda|}$. In this context, when v_j is ready to be scheduled at $s_{v_j}^{|\Lambda|}$, the core must be fully occupied to delay it. Hence, there must be $|\Lambda|$ number of nodes from $\mathcal{IB}_{v_j}^{final}$ to execute in parallel at $s_{v_j}^{|\Lambda|}$ to delay v_j. Otherwise, with work-conserving scheduling, v_j can always start on an empty core at $s_{v_j}^{|\Lambda|}$. □

Furthermore, not all nodes within $\mathcal{IB}_{v_j}^{final}$, despite being parallel, can delay v_j. The priority characteristics of the set of $|\Lambda|$ parallel nodes still need to be considered. To systematically address this interaction, we propose the following lemma:

Lemma 4. *For a given node $v_x \in \mathcal{IB}_{v_j}^{final}$, if $P_{v_x} < P_{v_j}$, and it aims to contribute to delaying v_j (i.e. $s'_{v_x} < s'_{v_j}$), the only feasible scenario is when $s'_{v_x} < f_{v_j}^{pre}$.*

Proof. We prove this by contradiction. Suppose for a low-priority node $v_x \in \mathcal{IB}_{v_j}^{final}$, the conditions $s'_{v_x} \geq f_{v_j}^{pre}$ and $s'_{v_x} < s'_{v_j}$ hold . This suggests that at the time point $f_{v_j}^{pre}$, when v_j is eligible to execute, the scheduler chooses to execute the lower-priority v_x instead. This scenario contradicts the highest-priority-first scheduling policy, where v_j should have been scheduled before v_x if both were ready. □

Based on Lemma 4, we propose the following lemma, which specifies the priority conditions for $|\Lambda|$ parallel nodes within $\mathcal{IB}_{v_j}^{final}$ must satisfy in order to potentially delay v_j. This further aids in precisely determining the scheduling interactions that could impact the timely execution of v_j.

Lemma 5. *For the $|\Lambda|$ parallel nodes in $\mathcal{IB}_{v_j}^{final}$ that can impose a delay on v_j, at least one of the nodes must satisfy $P_{v_x} > P_{v_j}$.*

Proof. According to Lemma 4, for nodes v_x with $P_{v_x} < P_{v_j}$ to contribute to a delay, they must execute concurrently with the latest finishing predecessor of v_j, satisfying $s'_{v_x} < f_{v_j}^{pre}$. Given that there are $|\Lambda|-1$ cores on which they can execute, a successful delay additionally requires at least one node to execute in the interval $f_{v_j}^{pre} < s'_{v_x} < s'_{v_j}$. Therefore, this node must have a higher priority than v_j ($P_{v_x} > P_{v_j}$) due to the scheduling rule "highest-priority-first", thereby utilising the full capacity of $|\Lambda|$ cores to create the delay. □

An example is presented in Example 5 to demonstrate the concepts of Lemmas 4 and 5.

Example 5. *As shown in Figure 5, assume that v_{pre} represents the predecessor workload of v_j. v_j is eligible to execute at $t = 3$, i.e., $f_{v_j}^{pre} = 3$, but is delayed until $t = 5$. The delaying nodes consist of two low-priority nodes (v_{low}) and one high-priority node (v_{high}).*

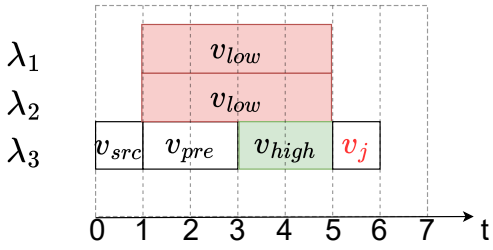


Fig. 5: An example to illustrate Lemmas 4 and 5.

The only plausible construction of the delay is that the two low-priority nodes start before $t = 3$ and occupy two cores, λ_1 and λ_2. The high-priority node starts at $t = 3$ on λ_3. Consequently, the delay extends v_j's start time from $t = 3$ to $t = 5$.

Lemma 5 represents the final step of the anomaly test which analyses whether v_j can finish later than its $f_{v_j}^{|\Lambda|}$ in any scenario. We build upon the analysis and propose the following theorem.

Theorem 1. *For a given DAG task τ_i, if all $v_j \in V_i$ pass the anomaly test in a scheduling scenario, the WCRT of τ_i in such a scenario can be bounded by simulating the long-makespan scenarios (i.e. calculating $f_{v_{sink}}^{|\Lambda|}$).*

Proof. If all $v_j \in V_i$ pass the anomaly test in a scheduling scenario. That means all nodes cannot finish later than their corresponding long-makespan scenario (i.e. $f'_{v_j} \leq f_{v_j}^{|\Lambda|}$) when there are nodes in the scheduling scenario that have $C'_{v_j} \leq C_{v_j}$. Hence, the timing anomaly does not exist in such a scheduling scenario, thus its WCRT can be bounded by the makespan of its long-makespan scenario. □

The process for conducting an anomaly test to determine the anomaly status of a scheduling scenario is as follows:

- First, Algorithm 1 is utilised to calculate the timing metrics of each node (say v_j), i.e., $f_{v_j}^{|\Lambda|}$ and $s_{v_j}^{|\Lambda|}$, which can be used to deduce $f_{v_j}^{pre}$ and the makespan of the scheduling scenario.
- Then, each node within the DAG is analysed to determine whether it has the potential to overrun its $f_{v_j}^{|\Lambda|}$.
- The problem is narrowed down to whether there will be any potentially additional delay to v_j if certain nodes finish earlier.
- The potential nodes where delays might occur – denoted as $\mathcal{IB}_{v_j}^{potential}$, $\mathcal{IB}_{v_j}^{remove}$, and $\mathcal{IB}_{v_j}^{final}$ in Equations (2), (3), and (5) – are progressively narrowed down and refined throughout the process.
- If the nodes in $\mathcal{IB}_{v_j}^{final}$ do not satisfy all conditions in Lemma 3, 4, and 5, then v_j will not overrun its $f_{v_j}^{|\Lambda|}$.
- If all nodes within a scheduling scenario are guaranteed not to overrun their $f_{v_j}^{|\Lambda|}$, then the entire scheduling scenario is deemed anomaly-free; otherwise, the scenario is considered anomalous.

With the proposed anomaly test through static analysis, some of the anomaly-free DAG task scheduling scenarios can be identified. At this stage, we can provide WCRT bounds for the anomaly-free scheduling through simulation. However, the anomaly test is a sufficient test which can only guarantee anomaly-free scheduling scenarios that pass the test, for those who fail to pass the test,

their anomaly status cannot be conclusively determined by static analysis.

4.3 Binary classification model

To comprehensively understand the anomaly status of all scheduling scenarios in a system, especially identifying those scheduling scenarios that cannot be directly determined by the anomaly test. We aim to utilise the machine-learning approach based on a wide range of scheduling data for different scheduling scenarios to train a model that can help to identify the anomaly status of scheduling scenarios. The complex nature and unpredictability of timing anomalies associated with DAG scheduling motivate such an approach.

As introduced in Section 4.2, with the knowledge of the DAG structure, WCET, and node priorities, the anomaly test can be used to directly identify anomaly-free task scheduling scenarios. Alternatively, if we run the system or its simulator long enough, we are likely to directly observe anomaly cases (i.e., a makespan longer than that of the simulated long-makespan scenario) in DAG task scheduling. The longer the system operates, the more comprehensive the data we can gather. In this context, we frame the problem as a binary classification task, where the input comprises scheduling scenario data, and the output indicates whether an anomaly is present in the schedule.

Once the binary data is acquired, it is straightforward to apply a supervised learning approach [38] to train a binary classification model. However, there are cases where a scheduling scenario remains undefined — neither passing the anomaly test nor being identified as an anomaly after extensive simulation. Training exclusively on easily identifiable data may result in a biased model [39]. To mitigate this, we adopt a semi-supervised recursive strategy: the model is first trained on known data and then used to predict labels for the undefined data. Predictions made with high confidence (e.g., 90%) are cyclically reintegrated into the training set, enabling continuous refinement of the model.

We select Random Forest for the semi-supervised training of our binary classification model because it effectively reduces overfitting and variance through its ensemble approach, resulting in stable and generalised models. Additionally, Random Forest provides better interpretability and robustness when dealing with limited labelled data, making it an ideal choice for our specific needs [40].

For the binary classification model, the output is straightforward: we define the output feature as *isAnomaly*, a binary indicator of whether an anomaly occurs in a given scheduling scenario. Next, we need to extract the input features from the collected data that represent a scheduling scenario and influence anomaly classification. The features we selected to reflect the varying execution times of a DAG across different releases are:

- *MaxMakespan* and *MinMakespan*: For a DAG task scheduled on a given number of cores with multiple releases, we select the maximum and minimum makespan observed.
- *MaxWorkload* and *MinWorkload*: The workload is the sum of execution times of nodes in a given release of a DAG task. With many releases observed, we find out the maximum and minimum workload.
- *MaxET*, *MinET*, and *MedianET*: For a given release of a DAG, we find out the node with maximum, minimum and median execution times. With many releases, we observe the maximum of these three values.

In terms of DAG structure, the following features are considered in modelling:

- *NumNodes*: The total number of nodes in the DAG.
- *MaxParallel*: The maximum number of nodes that can execute concurrently in the DAG.
- *CriticalPath*: The duration of the longest path through the DAG [6].
- *AvgInDegree* and *AvgOutDegree*: The average in-degree and out-degree per node refer respectively to the average number of incoming and outgoing directed edges connected to a node in a graph.

Additional relevant features include:

- *CoreNum*: The number of processing cores in a given scheduling scenario.

By transforming the collected data into extracted input and output features, we obtain a prepared dataset for model training. To address class imbalance, a balanced labelled dataset is constructed by ensuring equal representation of anomaly-free and anomalous cases. This labelled dataset is randomly split into a training set and a validation set, with the majority portion allocated for training.

For Random Forest training, the number of trees is empirically selected to balance model stability and computational efficiency. The training process is conducted exclusively on the training set, which initially contains only the labelled training data. A baseline model is first trained on this set. In each subsequent iteration, the model is used to predict labels for a separate pool of undefined scenarios reserved for semi-supervised training. Predictions with confidence scores exceeding a predefined threshold are incrementally added to the training set, and the model is retrained on the updated data. This iterative process continues until convergence.

After training is complete, the final model is evaluated using the validation set, which remains strictly isolated from the training process to ensure an unbiased assessment. Performance is measured using standard classification metrics derived from the confusion matrix, including Precision, Recall, F1-score, and Accuracy for each class which will be explained in Section 5.2.

Finally, the trained model is applied to a large set of previously unseen, unlabeled scheduling scenarios. For these scenarios, ground-truth labels are unavailable, making direct evaluation infeasible. Instead, we analyse the distribution of the model's predicted confidence scores across this dataset. This confidence distribution provides valuable insights into the model's behaviour and generalisation capability.

The integration of machine learning enhances the completeness of the proposed analysis approach. It enables the utilisation of operational data from the running system to predict the anomaly status of scheduling scenarios from a system that cannot be easily defined. The proposed hybrid approach, combining the anomaly test through static analysis with machine learning techniques, enables system engineers to comprehensively identify different scheduling scenarios and provide more specific WCRT bounds for DAG task scheduling.

4.4 An application of the hybrid approach

In embedded real-time systems, WCRT bounds are typically computed during the offline design phase, based on known task information such as DAG structures and task execution times. These bounds are essential for verifying timing correctness prior to system deployment. The proposed hybrid approach enhances this offline analysis by providing tighter WCRT bounds for scheduling

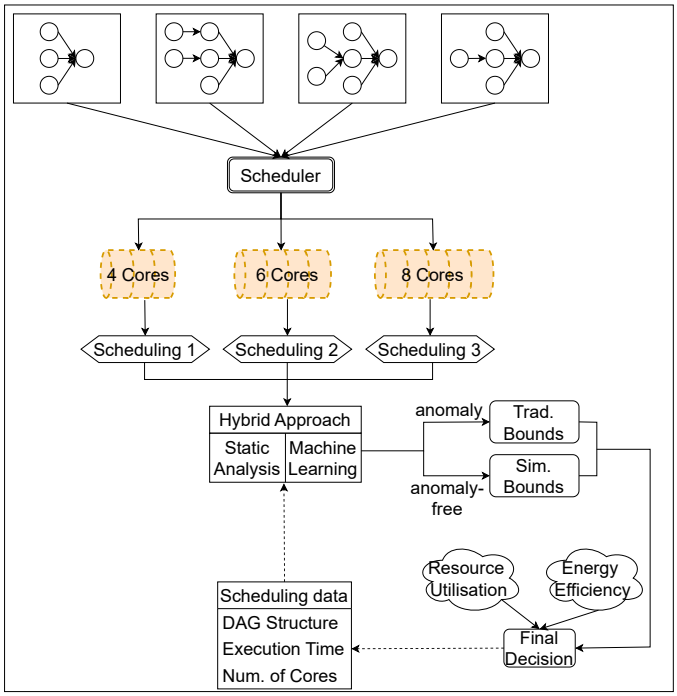


Fig. 6: An application of the hybrid approach

scenarios identified as anomaly-free. The approach is designed for offline use only and is intended to support the design and refinement of scheduling strategies.

During the design phase, a simulator is used to generate a variety of scheduling scenarios by varying system configurations, such as the number of available cores (e.g., 4, 6, or 8 cores, as illustrated in Figure 6). Each scenario is analysed by the hybrid approach to detect potential timing anomalies. For anomaly-free scenarios, the resulting simulated bounds can replace pessimistic conventional WCRT estimates [8], [26], enabling more efficient designs. Based on these offline comparisons, designers can make informed scheduling decisions. For instance, if a DAG task meets its deadline with four cores, the system can safely assign only four cores to that task, improving overall resource utilisation. For scenarios with potential anomalies, traditional WCRT bounds remain applicable to ensure timing safety.

The simulator-generated data are also used to train a binary classification model, as described in Section 4.3, to further assist in identifying scheduling scenarios. Both the training of this model and the computation of hybrid bounds are performed offline. After system deployment, additional data may be collected from the running system to further improve the classification model. This additional training remains offline and does not affect the real-time behaviour of the system. The only online cost is data gathering, which does not affect scheduling performance. Throughout this process, the hybrid approach targets providing tighter offline WCRT bounds to guide system design, and its timing complexity is not a primary concern, as it does not impact runtime system performance.

5 Evaluation

In this section, we first describe the data generation of DAG scheduling scenarios by the simulator we develop. We then detail the training of a binary classification model. Additionally, we evaluate the effectiveness of the proposed static analysis approach and WCRT bounds.

5.1 The generation of DAG scheduling scenarios

Before starting the training, we first develop a DAG task generator that manages the task's scale through two structural dimensions: the number of node layers, called *Length* and the number of nodes within each layer, termed *Parallelism*. The process begins with a single source node and incrementally adds nodes based on the specified *Length* and *Parallelism*. It concludes with a singular sink node. Dependencies are constructed with a 50% chance of connecting nodes to those in the preceding layer. To ensure cohesiveness, nodes without predecessors or successors are linked to the source or sink node. It is worth noting that *Parallelism* and *Length* generally control the vertical and horizontal scale of a DAG, it does not strictly determine the number of parallel nodes or the critical path length of a given DAG due to the random generation of dependencies between nodes. This approach effectively controls the DAG's scale while enabling random connections within the same scale.

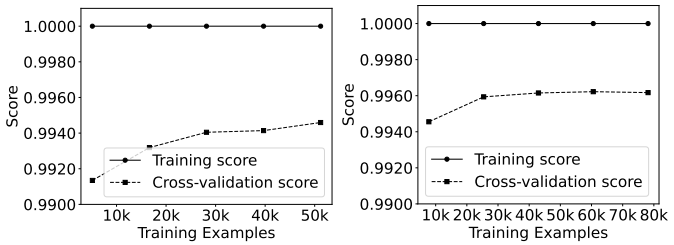
We then generate a series of DAG scheduling scenarios containing DAG tasks with varied *Length* $\in [4, 15]$ and *Parallelism* $\in [4, 15]$ running on a set of identical *Core* $\in [2, 16]$. For each setting, we generate 1,000 DAG tasks. The period of each DAG τ_i is randomly determined within the range of $T_i \in [1000ms, 3000ms]$. Since the system schedules one DAG task on dedicated cores at a time, utilisation is not critical as long as it is within a reasonable range. The utilisation of a task is then set to $U_{\tau_i} = 0.5$. With utilisation determined, the total WCET of τ_i is calculated by $C_{\tau_i} = U_{\tau_i} \times T_i$. The WCET of each node (e.g., C_{v_j} for a node v_j) is then randomly distributed by C_{τ_i}, where we enforce $C_{v_j} > 0$. A unified node-level priority assignment algorithm is applied to each DAG which follows the concepts of front layer first and highest WCET first.

Each scheduling scenario is analysed through the anomaly test described in Section 4.2. The scheduling scenarios that pass the anomaly test are identified as anomaly-free and labelled '0'. For the remaining scenarios, we use Algorithm 1 to simulate the makespan of scheduling scenario 100,000 times. Each time, the execution time of each node within a DAG is randomly varied from 50% to 100% of its WCET. If any anomaly case (makespan longer than that in the long-makespan scenario) is identified out of the 100,000 simulations we label the scenario with '1'. This approach imitates the execution variations that a DAG task can incur in a scheduling scenario with different releases. For those scenarios that are unable to identify as either '0' or '1', we denote them as undefined scheduling scenarios. For other features of a scheduling scenario denoted in Section 4.3, the *MaxWorkload* and *MaxMakespan* are generated using 100% of the WCET of each node. The *MinWorkload* and *MinMakespan* are generated using 50% of the WCET of each node. The *MaxET*, *MinET*, and *MedianET* are the maximum, minimum, and median of 100% WCET of nodes within a DAG, respectively.

5.2 Binary classification evaluation

To train the model specified in Section 4.3, we randomly select 40,000 scheduling scenarios labelled with '1' (anomaly) and '0' (anomaly-free) respectively out of the generated data which prepares a balanced dataset for training. Moreover, we further collect 80,000 undefined scheduling scenarios for performing the

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674



(a) (First) Initial data. (b) (Second) Inclusion of new data.

Fig. 7: Learning curves on training data.

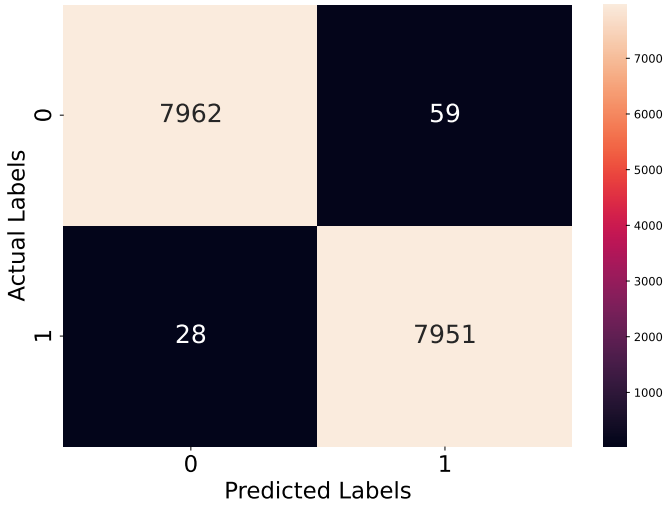


Fig. 8: Confusion matrix for validation data.

TABLE 2: Classification report.

Class	Precision	Recall	F1-score	Support	Accuracy
0	1.00	0.99	0.99	8021	0.995
1	0.99	1.00	0.99	7979	
Macro Avg	0.99	0.99	0.99	16000	
Weighted Avg	0.99	0.99	0.99	16000	

semi-supervised training. As introduced in Section 4.3, the inputs are features that describe the scenarios, and the output is the labels ('0' or '1'). The combined 80,000 labelled dataset is split into 80% of the training set and 20% of the validation set which is a standard practice.

Our Random Forest classifier, initialised with 100 trees, uses a semi-supervised training approach. It starts with labelled data, then predicts probabilities on the undefined dataset. Instances with predictions above a 90% confidence threshold are added to the training set as correctly labelled. This iterative process continues for up to five retraining cycles or until no further high-confidence instances or accuracy improvements are found, effectively avoiding model bias.

Our model underwent two training rounds, achieving high accuracy in the first round. In the second round, we incorporated 33,678 additional high-confidence instances from the undefined dataset to potentially enhance the model's performance. Despite this, the lack of significant improvement in the second round prompted us to terminate further training.

Figure 7 shows the learning curves for both training rounds. The graphs display accuracy scores plotted against the number of training examples, with separate lines for training and cross-validation scores, generated using 5-fold cross-validation on the training data (the 80%). The number of training examples increases gradually. For each plot in the figure, the training examples are split into 5 parts. The model is trained on 4 parts and validated on the remaining part. This process is repeated 5 times, with each fold serving as the validation set once. As for the 'Training Score', the model's performance is evaluated on the same 4 folds it was trained on, resulting in 5 training scores. The plot shows the average of these 5 training scores, indicating how well the model fits the data it has just seen. For the 'Validation Score', the plot illustrates the average of the 5 validation scores. Figure 7b shows the learning curve for the second round, which includes newly predicted data from the undefined dataset that meets the threshold. As a result, it contains more training examples

In both rounds, the training score remains consistently close to 1.0, indicating strong fitting capacity and minimal bias. In the first round, the cross-validation score starts near 0.992 and gradually rises to around 0.9945, suggesting the model generalises well as more labelled data is introduced. After incorporating 33,678 high-confidence pseudo-labelled samples in the second round, the cross-validation score improves further, stabilising around 0.996. This indicates a modest but clear benefit from the inclusion of additional data.

Compared to the first round, the learning curve in the second round shows a slower rise and a smoother convergence, reflecting the increased sample size and potential label noise from the pseudo-labelled data. Nonetheless, the generalisation gap decreases, suggesting enhanced robustness. Given that both rounds reach a high level of stable accuracy, we conclude that the model already performed effectively after the first round, and the second round provides only marginal gains. Therefore, subsequent analyses primarily reference the results from the first training round.

The confusion matrix in Figure 8 shows the alignment between the model's predictions and actual outcomes conducted on the validation data (the 20%). The rows represent predicted classes, while the columns represent actual classes. The matrix indicates that the model correctly classified 7,962 instances as class 0 and 7,951 as class 1, representing true negatives (TN) and true positives (TP), respectively. It also reveals 28 false positives (FP) and 59 false negatives (FN). The high number of correct predictions and low number of errors suggest that the model effectively distinguishes between the classes.

Table 2 presents key classification metrics derived from the confusion matrix, including *Precision*, *Recall*, *F1-score*, and overall *Accuracy*. These metrics are computed based on the standard definitions: *TP*, *FP*, *TN*, and *FN*.

The model achieves a high *Accuracy* of 0.995, indicating that 99.5% of instances in the validation set are correctly classified. Class-wise *Precision* and *Recall* are both above 0.99, demonstrating that the classifier commits very few false positives and false negatives. Specifically, *Precision* $(\mathrm{TP}/(\mathrm{TP}+\mathrm{FP}))$ quantifies the proportion of predicted positives that are correct, while *Recall* $(\mathrm{TP}/(\mathrm{TP}+\mathrm{FN}))$ measures the proportion of actual positives that are correctly identified.

The *F1-score*, calculated as the harmonic mean of Precision and Recall, reaches 0.99 for both classes, confirming the model's balanced performance across sensitivity and specificity. The nearly equal support values (8021 for Class 0 and 7979 for Class 1)

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674

reflect a well-balanced dataset, which contributes to stable performance. Moreover, the high *Macro* and *Weighted* average scores further affirm the model's robustness and its consistent behavior across both classes, regardless of label distribution. Overall, these metrics provide strong evidence of the classifier's reliability and generalisation capability.

Furthermore, we used the entire training dataset to generate the feature importance chart for the Random Forest classifier, shown in Figure 9, highlights the factors affecting the model's accuracy of predictions. Feature importance in a Random Forest model is determined by measuring how significantly each feature reduces impurity (such as Gini impurity or entropy for classification) in splits across all trees. The importance is quantified by averaging these impurity reductions for each feature throughout the forest, thus indicating its contribution to the model's accuracy [41].

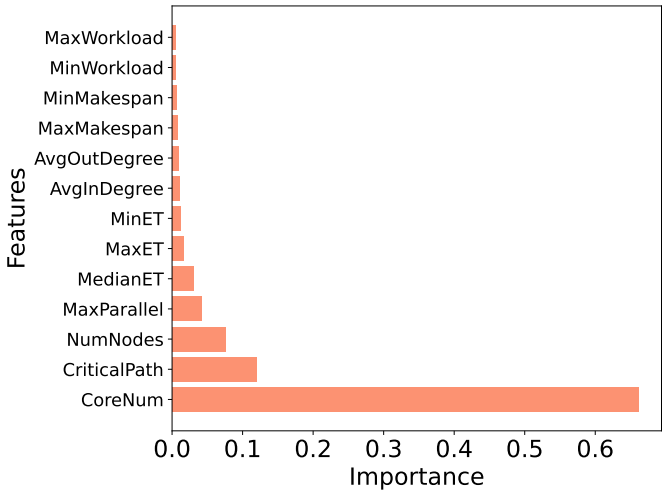
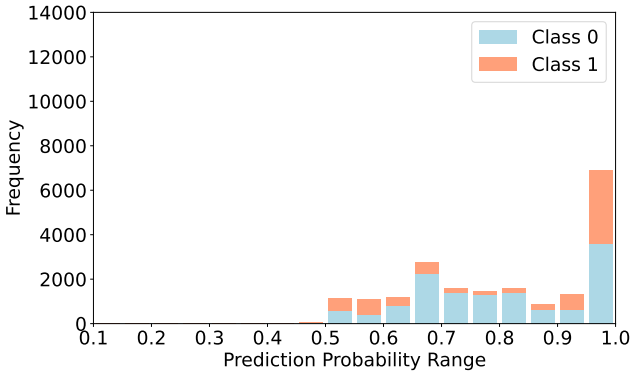


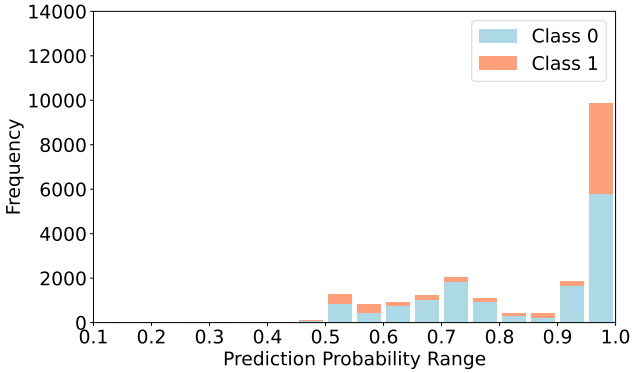
Fig. 9: Feature importance of all input features.

The feature importance plot in Figure 9 reveals that *CoreNum* is by far the most influential feature, with an importance score exceeding 0.6. This dominance suggests a strong correlation between the number of available cores and the classification of anomalies. As explained in Lemma 2, anomalies in scheduling scenarios often result from changes in execution order. Since the number of cores determines the level of parallelism in task execution, it directly affects such interactions and, consequently, the anomaly outcome. *CriticalPath* and *NumNodes* follow as the next most important features, albeit with substantially lower importance scores. This is intuitive, as they influence the structural length and size of the DAG, which indirectly affects scheduling complexity. *MaxParallel* and *MedianET* also contribute modestly to the model's decisions, indicating that execution characteristics still have some role. In contrast, features such as *AvgInDegree*, *AvgOutDegree*, *MinMakespan*, *MinWorkload*, and *MaxWorkload* show minimal importance. These results suggest that while workload and timing variability influence scheduling to a degree, they are secondary to the structural and resource-based determinants like *CoreNum*. Overall, the feature importance analysis provides a clear rationale for model performance and can guide future feature selection strategies by highlighting which features are truly influential in predicting anomalies.

At this stage, we evaluate the effectiveness of the trained binary classification model by applying it to 20,000 previously unseen scheduling scenarios without ground-truth labels. These



(a) First-round model



(b) Second-round model

Fig. 10: The confidence distribution on unlabelled data

samples were not used during training and represent cases that the anomaly test alone could not resolve, as discussed in Section 4.3.

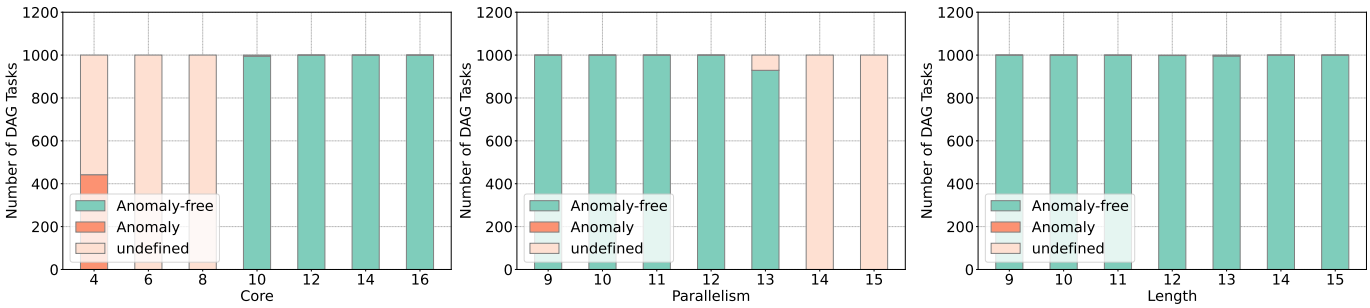
Figure 10 illustrates the distribution of the model's prediction confidence for these unlabelled samples, comparing the outputs from the first-round and second-round models in the semi-supervised learning process. Both subplots show a strong skew toward higher confidence predictions, particularly in the range $[0.9, 1.0]$, indicating that the models are generally confident in their outputs. The first-round model (Figure 10a) displays a broader spread of predictions across the intermediate confidence range $[0.5, 0.8]$. In contrast, the second-round model (Figure 10b) shows a pronounced concentration in the highest confidence bin, suggesting an increase in certainty. This improvement is likely a result of incorporating high-confidence pseudo-labelled data during retraining, which helped the model better capture the decision boundaries in the unlabelled data.

It is also evident that both models are more inclined to predict Class 0 with higher confidence. This trend may reflect the underlying class distribution in the sampled 20,000 scenarios, which could contain a greater proportion of Class 0 instances. Overall, the second-round model demonstrates enhanced confidence and better adaptation to the unlabelled data, validating the utility of the semi-supervised learning approach in this context.

5.3 WCRT bounds evaluation

As shown in Figure 9, the top four relevant features are *CoreNum*, *CriticalPath*, *NumNodes*, and *MaxParallel*. These features can be controlled by scheduling scenarios generation settings: *Core*, *Parallelism*, and *Length*. As shown in Figure 11, we select scheduling

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2025.3603674



(a) Parallelism = 10, Length = 8 (b) Core = 12, Length = 8 (c) Core = 12, Parallelism = 12

Fig. 11: The distribution of scheduling scenarios across different settings.

TABLE 3: Comparison of WCRT bounds with existing methods under varying parameters.

Parameter Value	Counts	MeanShorter [8]	MaxShorter [8]	MeanShorter [28]	MaxShorter [28]	MeanShorter [4]	MaxShorter [4]
(Parallelism = 10, Length = 8) Core =							
10	995	16.12%	52.11%	28.92%	54.69%	27.33%	45.09%
12	1000	2.71%	36.21%	23.43%	42.64%	22.70%	36.27%
14	1000	0.08%	16.48%	19.70%	33.24%	19.36%	29.08%
16	1000	0.00%	0.00%	17.15%	30.72%	17.01%	29.59%
(Core = 12, Length = 8) Parallelism =							
9	1000	0.43%	23.00%	21.08%	35.37%	20.66%	31.15%
10	1000	2.71%	36.21%	23.43%	42.64%	22.70%	36.27%
11	1000	6.99%	40.43%	25.72%	48.15%	24.61%	39.16%
12	1000	15.42%	47.66%	28.16%	62.65%	26.51%	40.79%
13	929	21.58%	55.47%	30.47%	54.14%	28.20%	49.07%
(Core = 12, Length = 12) Length =							
9	1000	15.98%	43.42%	28.53%	45.22%	26.99%	38.42%
10	1000	15.94%	44.12%	28.64%	42.21%	27.18%	38.52%
11	999	16.63%	44.90%	29.02%	50.97%	27.57%	40.31%
12	998	16.61%	37.70%	29.19%	44.02%	27.76%	40.69%
13	996	16.61%	42.55%	29.17%	43.07%	27.82%	38.03%
14	1000	17.41%	40.94%	29.54%	40.20%	28.12%	38.72%
15	1000	17.29%	43.21%	29.58%	42.49%	28.20%	39.75%

scenarios under various settings of *Core*, *Parallelism*, and *Length*, and categorised them as *anomaly*, *anomaly-free*, or *undefined*, to examine anomaly behaviours across different settings. The generation of labels is explained in Section 5.1.

As shown in Figure 11, the three sets of bar charts illustrate the distribution of scheduling scenarios labelled as anomaly-free, anomaly, or undefined under various settings, each containing 1,000 scheduling scenarios.

Figure 11a displays the labelling of scheduling scenarios across different *Core* counts, ranging from 4 to 16, while keeping *Parallelism* fixed at 10 and *Length* at 8. Most scheduling scenarios are labelled as anomaly-free when $Core \geq 10$. In contrast, lower *Core* counts, particularly between 4 and 8, result in a significant proportion of scenarios labelled as anomaly or undefined. This suggests that when $Core < Parallelism$, anomalies are likely to occur. When $Core \geq Parallelism$, most scheduling scenarios are anomaly-free. This occurs because when $Core \geq Parallelism$, nodes can execute in parallel, reducing contention and unpredictability, thereby minimising anomalies.

A similar phenomenon is observed in Figure 11b, which shows the distribution of labels across varying levels of *Parallelism*, from 9 to 15, while keeping *Core* and *Length* constant at 12 and 8. The number of anomaly-free scheduling scenarios decreases as *Parallelism* increases, while the number of anomaly or undefined scheduling scenarios rises. This trend further supports the observation that scheduling scenarios are likely to be anomaly-free when $Core \geq Parallelism$. Additionally, there are many undefined scheduling scenarios when $Parallelism \geq 14$. These scheduling scenarios do not pass the anomaly test, and after 100,000 rounds of simulation, they are still challenging to identify. In this case, the model trained in Section 5.2 can help identify their anomaly status, enabling engineers to select a more precise WCRT solution and improve system performance.

Following the observations from Figures 11a and 11b, we set $Core = Parallelism = 12$ while increasing *Length* from 9 to 15. As illustrated in Figure 11c, most scheduling scenarios can be identified as anomaly-free when $Core = Parallelism$, even as the scale of the DAG task increases in *Length*. At this stage of observation, from a system design perspective, tighter WCRT bounds are achievable for scheduling scenarios when DAG tasks are long, and $Core \geq Parallelism$.

For the anomaly-free scheduling scenarios, we provide their WCRT based on the long-makespan scenario computed by Algorithm 1, which is proven to be safe in Section 4.2. We then compare the proposed WCRT bound for each scheduling scenario against existing bounds reported in [4], [8], [28], with the results summarised in Table 3 to prove the effectiveness of the proposed approach. Among these, [8] and [4] represent the most recent works that specifically target global work-conserving scheduling under a limited preemption scheme. The bound in [28] is also comparable with the inclusion of limited pre-emption features. In contrast, [6], [29] contain unresolved flaws in their analysis and are therefore excluded from the comparison. Other existing algorithms are based on substantially different system models or

scheduling assumptions and are thus not considered comparable.

Table 3 consists of three parts, corresponding to the three subfigures in Figure 11, and presents the performance of the proposed WCRT bound when a scenario is determined to be anomaly-free by the proposed anomaly test. The first column indicates the parameter values of *Length*, *Core*, and *Parallelism*. The second column *Counts* represents the number of anomaly-free scheduling scenarios under each setting. The columns *MeanShorter* and *MaxShorter* show the average and maximum percentage reductions of our simulated WCRT compared to the existing methods for the same anomaly-free scheduling scenarios.

The first part of Table 3 corresponds to the anomaly-free cases shown in Figure 11a, where the number of *Cores* ranges from 10 to 16 while keeping *Length* = 8 and *Parallelism* = 10 constant. Compared to [8], when *Core* = 10, 995 (out of 1000) anomaly-free scheduling scenarios exhibit an average reduction of 16.12% in WCRT bounds, with a maximum reduction of 52.11%. As the number of *Cores* increases, this advantage gradually decreases. This trend occurs because the bound in [8] does not include much pessimism when ample cores are available for scheduling. However, since the proposed simulated bound is an exact bound, it remains strictly tighter than any static analysis bound. A similar trend is observed in the second part of the table, which corresponds to Figure 11b. Here, *Parallelism* varies while keeping *Core* = 12 and *Length* = 8 constant. When *Parallelism* = 9, the improvement is modest at 0.43%. However, increasing *Parallelism* to 13 yields an average advantage of 21.58% across 929 scheduling scenarios, with a maximum reduction of 55.47%.

The third part of the table, corresponding to Figure 11c, focuses on varying *Length* from 9 to 15 while keeping *Parallelism* = *Core* = 12. In this case, the average improvement consistently exceeds 15%. Notably, when *Length* = 15, a maximum reduction of 43.21% is observed, and the average improvement reaches 17.29%. Compared to prior approaches such as [4], [28], the proposed simulated bound consistently yields shorter WCRT estimates—typically between 20% and 30% across all settings—demonstrating its effectiveness and superiority over existing methods. Overall, Table 3 indicates that a substantial number of scheduling scenarios can be identified as anomaly-free through static analysis alone. Once a scenario is confirmed to be anomaly-free, the proposed simulated bound provides significantly tighter estimates than the state-of-the-art. This direct reduction in the WCRT bound for DAG scheduling represents a substantial advancement [6], [28].

The evaluation results show that the proposed hybrid approach is effective. The anomaly detection through static analysis can directly identify anomaly-free scheduling scenarios and provide tighter bounds. For other scheduling scenarios, the trained model can predict anomalies with high accuracy. It is important to note that the model used here is only intended to demonstrate the applicability of the hybrid approach and guide how machine learning can be applied to this type of problem. This is not a universal solution; different data or machine learning models can be explored. If a scheduling scenario is predicted to have an anomaly, traditional bounds can still be applied for safety. For scenarios predicted to be anomaly-free, since the model's predictions are not fully accurate, the decision on which bounds to use should be made by the system engineer based on the system's safety requirements.

6 Conclusion

In conclusion, this paper proposes a novel hybrid approach to analyse the anomaly status of different DAG scheduling scenarios and provide tighter WCRT bounds. The constructed anomaly test through static analysis can be used to identify some anomaly-free scheduling scenarios directly. Then, a binary classification model is trained on tested anomaly-free cases and observed anomaly cases to predict the anomaly status of undefined scheduling scenarios with an accuracy of 99.5%. This hybrid approach provides a comprehensive understanding of the system, enabling significantly tighter WCRT bounds—up to an average improvement of 21.58% in anomaly-free scheduling scenarios and a maximum reduction of up to 55.47%. This is the first work to provide a timing guarantee for real-time systems by combining static analysis with machine learning. In the future, we will extend this work to a more complex system such as mixed-criticality multi-DAG systems, where multiple DAG tasks are scheduled concurrently and with different criticality levels.

References

[1] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, 2016.

[2] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A definition and classification of timing anomalies," in *International Workshop on Worst-Case Execution Time Analysis*, 2006.

[3] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.

[4] M. A. Serrano, A. Melani, M. Bertogna, and E. Quiñones, "Response-time analysis of DAG tasks under fixed priority scheduling with limited preemptions," in *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[5] Q. He, N. Guan, Z. Guo *et al.*, "Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2019.

[6] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[7] J. Abella, C. Hernández, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega, "WCET analysis methods: Pitfalls and challenges on their trustworthiness," in *IEEE International Symposium on Industrial Embedded Systems*, 2015.

[8] N. Chen, S. Zhao, I. Gray, A. Burns, S. Ji, and W. Chang, "Precise response time analysis for multiple DAG tasks with intra-task priority assignment," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023.

[9] P. Voudouris, P. Stenström, and R. Pathan, "Timing-anomaly free dynamic scheduling of task-based parallel applications," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017.

[10] P. Chen, W. Liu, X. Jiang, Q. He, and N. Guan, "Timing-anomaly free dynamic scheduling of conditional DAG tasks on multi-core systems," *ACM Transactions on Embedded Computing Systems*, 2019.

[11] G. Dai, M. Mohaqeqi, and W. Yi, "Timing-anomaly free dynamic scheduling of periodic DAG tasks with non-preemptive nodes," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2021.

[12] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response time analysis of sporadic DAG tasks under partitioned scheduling," in *IEEE Symposium on Industrial Embedded Systems (SIES)*, 2016.

[13] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *IEEE Real-Time Systems Symposium (RTSS)*, 2018.

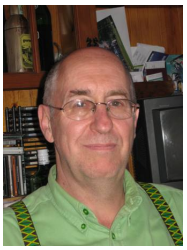
[14] S. Baruah, "Scheduling DAGs when processor assignments are specified," in *International Conference on Real-Time Networks and Systems*, 2020.

[15] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[16] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.
[17] M. Qamhieh, L. George, and S. Midonnet, "A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems," in *International Conference on Real-Time Networks and Systems*, 2014.
[18] S. Baruah, "Federated scheduling of sporadic DAG task systems," in *IEEE International Parallel and Distributed Processing Symposium*, 2015.
[19] ——, "The federated scheduling of systems of conditional sporadic DAG tasks," in *IEEE International Conference on Embedded Software (EMSOFT)*, 2015.
[20] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela, "The global EDF scheduling of systems of conditional sporadic DAG tasks," in *IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.
[21] K. Wang, X. Jiang, N. Guan, D. Liu, W. Liu, and Q. Deng, "Real-time scheduling of DAG tasks with arbitrary deadlines," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2019.
[22] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global fp scheduling," in *International Conference on Real-Time Networks and Systems*, 2017.
[23] ——, "Schedulability analysis of DAG tasks with arbitrary deadlines under global fixed-priority scheduling," *Springer Real-Time Systems*, 2019.
[24] Q. He, N. Guan, M. Lv, X. Jiang, and W. Chang, "Bounding the response time of DAG tasks using long paths," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022.
[25] ——, "The shape of a DAG: bounding the response time using long paths," *Springer Real-Time Systems*, 2024.
[26] Q. He, N. Guan, S. Zhao, and M. Lv, "Multi-path bound for DAG tasks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.
[27] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *31st Conference on Real-Time Systems*, 2019.
[28] Q. He, M. Lv, and N. Guan, "Response time bounds for DAG tasks with arbitrary intra-task priority assignment," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2021.
[29] S. Zhao, X. Dai, and I. Bate, "DAG scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, 2022.
[30] Q. He, J. Sun, N. Guan, M. Lv, and Z. Sun, "Real-time scheduling of conditional DAG tasks with intra-task priority assignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
[31] J. Sun, F. Li, N. Guan, W. Zhu, M. Xiang, Z. Guo, and W. Yi, "On computing exact wcrt for DAG tasks," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.
[32] R. Bi, Q. He, J. Sun, Z. Sun, Z. Guo, N. Guan, and G. Tan, "Response time analysis for prioritized DAG task with mutually exclusive vertices," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022.
[33] H. Liang, X. Jiang, N. Guan, Q. He, and W. Yi, "Response time analysis and optimization of DAG tasks exploiting mutually exclusive execution," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.
[34] M. Verucchi, I. S. Olmedo, and M. Bertogna, "A survey on real-time DAG scheduling, revisiting the global-partitioned infinity war," *Springer Real-Time Systems*, 2023.
[35] A. JONES, L. RABELO, and Y. YIH, "A hybrid approach for real-time sequencing and scheduling," *International Journal of Computer Integrated Manufacturing*, 1995.
[36] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *The seventh ACM international conference on Embedded software*, 2009.
[37] F. Reghenzani, L. Santinelli, and W. Fornaciari, "Dealing with uncertainty in pWCET estimations," *ACM Transactions on Embedded Computing Systems (TECS)*, 2020.
[38] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Springer Machine learning techniques for multimedia: case studies on organization and retrieval*, 2008.
[39] B. Chen, J. Jiang, X. Wang, P. Wan, J. Wang, and M. Long, "Debiased self-training for semi-supervised learning," *Advances in Neural Information Processing Systems*, 2022.
[40] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines," *Elsevier Ore Geology Reviews*, 2015.
[41] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2009.

Nan Chen is a research fellow in the Computer Science Department at the University of York. He received his PhD in Computer Science in 2024 and his BEng in Electronic Engineering in 2019, both from the University of York, UK. His research interests include DAG task scheduling, reliable resource sharing, and mixed-criticality systems.

Xiaotian Dai is a Lecturer (Assistant Professor) in the Real-Time and Distributed Systems group. He received his PhD in 2019 from the University of York (Best Thesis) and holds an MSc in Control Systems from the University of Sheffield and a BSc in Control Engineering. His research focuses on timing analysis, task scheduling for multi-core systems, and assurance of timing in autonomous and cyber-physical systems.



Alan Burns (Fellow, IEEE) is a professor in the Department of Computer Science at the University of York, UK. He is a fellow of the Royal Academy of Engineering and has served as chair of the IEEE Technical Committee on Real-Time Systems. He has published over 500 papers and 10 books.

Iain Bate is a professor in Real-Time Systems within the Computer Science Department at the University of York. His research interests include scheduling, timing analysis, and the design and certification of Cyber-Physical Systems. He has chaired three leading international conferences and served on many programme committees. He was Editor-in-Chief of the Microprocessors and Microsystems journal and the Journal of Systems Architecture for 15 years, and was a Visiting Professor at Mälardalen University, Sweden, for five years. He is a member of the IEEE.