International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 5, pp 651-667, May-2026

https://doi.org/10.47001/IRJIET/2026.105088

Explainable AI for CPU Scheduling Under Real-Time Constraints

[1]I. Janidu Chinthana, [2]Samantha Rajapaksha

[1]Faculty of Graduate Studies, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka

[2]Professor, Department of Information Technology, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka

E-mail: [1]ms25903188@my.sliit.lk, [2]samantha.r@sliit.lk

Abstract **- Explainable AI for scheduling has advanced most in cloud, cluster, and production settings, where explanation is used to interpret learned policies, support operator trust, and improve policy tuning [1]–[4]. Yet these settings rarely face the timing constraints of OS-level CPU scheduling, where even small overheads can perturb dispatch behavior, wakeup latency, and deadline satisfaction [5]–[7]. This review examines the impact of XAI on CPU scheduler behavior and usability, with a focus on the unresolved problem of real-time XAI at OS timescales. The review synthesizes literature from five connected areas: explainability for learned schedulers, machine learning in OS scheduling, programmable Linux scheduling substrates, explainable schedulability analysis, and low-impact tracing for real-time diagnosis [5], [6], [8]–[10]. It argues that current work contains the ingredients for real-time XAI, but not yet an integrated solution that jointly evaluates explanation fidelity, timing safety, and debugging value under microsecond-scale overhead limits. A fixed-priority real-time scheduler is used as the anchor case study because it exposes the strongest contrast between learned priority selection and explainable timing evidence [9], [11], [12]. The analysis shows that the most viable form of XAI for CPU scheduling is not heavyweight local attribution in the kernel fast path, but a layered design combining lightweight runtime evidence, trace-based diagnosis, and certificate-style schedulability support [6], [9], [10]. The review concludes by proposing a metric framework for real-time XAI that ties latency, jitter, deadline misses, explanation fidelity, and debugging value together. This positions real-time XAI for CPU scheduling as a distinct research problem at the boundary of interpretability, operating systems, and real-time systems.**

Keywords: Explainable AI, CPU scheduling, Real-time systems, Operating systems, Fixed-priority scheduling, Schedulability analysis, Linux scheduling.

I. INTRODUCTION

Explainable AI for scheduling has grown fastest in clusters, cloud systems, and production planning, where explanation is usually applied to learned policies through surrogate rules, feature attribution, or policy distillation [1]–[4]. In those settings, explanation is mainly used to improve trust, reveal hidden policy biases, and support tuning by operators [1], [3]. The cost of generating an explanation is often not part of the scheduler hot path.

CPU scheduling inside an operating system is a harder setting. Kernel schedulers act at microsecond to low millisecond scales, and even modest extra work can alter wakeup latency, migration behavior, cache locality, and deadline miss risk [5]–[7]. This creates a gap between the current XAI literature and the needs of OS scheduling. Real-time XAI is already recognized as a broader open problem for time-critical systems [13]–[15]. In OS kernels, that gap becomes extreme because explanation overhead can perturb the behavior being explained.

The literature now shows two strong but mostly separate lines of progress. One line shows that machine learning can be brought close to the scheduler fast path with careful sensing and lightweight inference [5], [16], [17]. The other line shows that real-time scheduling can be made explainable through schedulability certificates, response-time analysis, and tracebased evidence [6], [9], [10]. What is still missing is an integrated account of how XAI affects scheduler behavior and usability when explanations must respect OS-level timing budgets.

This review focuses on that gap. The core claim is that XAI for CPU scheduling is useful only if explanation quality, timing impact, and developer usability are evaluated together. The review centers the discussion on fixed-priority real-time scheduling because it provides the clearest bridge between learned policy design and explainable timing evidence [9], [11], [12].

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

The main contributions of this review are threefold:

- It brings together literatures that are usually treated separately: explainable scheduling, learned OS policies, real-time schedulability evidence, and low-impact timing diagnosis [5], [6], [9].
- It argues that OS-level real-time XAI is a distinct problem because explanation must fit within microsecond-scale scheduling budgets [5], [7].
- It proposes a concrete evaluation frame that ties latency, jitter, deadline misses, explanation fidelity, and debugging value together [3], [6], [9].

II. METHODS

The review is organized around a project library built from a deep literature search on explainable AI for CPU and task scheduling, with emphasis on OS-level schedulers and realtime constraints. The most relevant papers cluster into six groups.

- Explainability methods for learned schedulers in clusters, clouds, and HPC [1]–[4]
- OS-level and kernel-adjacent ML scheduling frameworks [5], [8], [17], [18]
- Programmable scheduling substrates for low-overhead experimentation [8], [18], [19]
- Explainable schedulability and timing evidence in realtime systems [7], [9], [10]
- Low-impact tracing and monitoring for latency and fairness diagnosis [6], [20]–[23]
- Position papers and surveys on learned OS policies and real-time AI constraints [14], [15], [24], [25]

The review uses these clusters to answer four questions.

- Where has XAI already improved scheduler behavior and usability?
- Why do those gains not transfer cleanly to OS scheduling?
- Why fixed-priority real-time scheduling is a strong anchor case?
- Which metrics are needed to evaluate real-time XAI credibly?

III. RELATED WORK

Existing review and position papers touch parts of this space, but none fully center the combination of XAI, CPU scheduling, and OS-level real-time constraints.

A. Explainability for learned schedulers outside the OS kernel

The most mature XAI work on scheduling comes from cloud, cluster, HPC, and production settings [1]–[4], [26]–[28]. These papers are important because they establish that explanation can do more than make a scheduler look transparent. It can expose brittle policy structure, reveal hidden feature dependence, and help operators tune or reject a learned policy.

The strongest examples are the DRL scheduling papers by Zhang et al. and Li et al. [1], [2]. Zhang et al. decompose a learned scheduler into job-level and task-level explanation layers, which is valuable because it recognizes that schedulers act on structured state rather than flat feature vectors [1]. Li et al. go further toward deployment by distilling a DRL scheduler into a decision-tree surrogate, making the policy easier to inspect and tune while retaining competitive performance [2]. These two papers are among the clearest demonstrations that interpretability can change scheduler usability by turning an opaque controller into something closer to an operational policy.

At the same time, this literature has clear limits. Fischer et al. show that common XAI techniques in production scheduling often lack falsifiability, stable terminology, and domaingrounded validation [3]. Gal´an et al. similarly show that even rule-based or fuzzy systems can lose interpretability once automatic learning pushes them toward performance objectives [27]. Ates et al. argue from the HPC side that explainability matters most when it helps debugging and trust under real telemetry, not when it merely produces visually plausible explanations [28]. Erlenbusch and Stricker's systematic review confirms that explainable reinforcement learning for scheduling is still methodologically fragmented [26]. The key point is that this literature is strong on human-facing explanation, but it typically assumes that explanation cost is not itself part of the scheduling problem.

That assumption is exactly what breaks in OS scheduling. Cloud and HPC schedulers usually operate at timescales where richer post hoc analysis is acceptable. CPU scheduling in kernels does not.

B. Learned OS scheduling and programmable scheduler substrates

A second literature study learned scheduling in operating systems, often without making explainability central [5], [8], [17]–[19], [29]–[35]. This body of work is crucial because it shows that sophisticated policy logic can be brought closer to the kernel and to user-space scheduling paths than older systems' work assumed.

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

The Linux kernel load balancing work by Chen et al. is especially important because it gives a concrete overhead reference point. Their in-kernel learned model reproduces migration decisions with high accuracy while adding only 1.9 microseconds of latency [5]. This does not solve explainability, but it establishes the scale of what is practically tolerable. Yang et al. similarly use eBPF feature collection and XGBoost-based priority prediction to reduce context-switchrelated overhead, though the reported gains are modest and the paper focuses on optimization rather than explanation [30]. Wang et al. push adaptation further by using a learned router that switches between expert schedulers at runtime through sched ext, showing that adaptive policy selection can be practical in Linux-like environments [31]. Kahu's KernelOracle is related in a different way. It models Linux scheduler behavior predictively, which is useful for understanding and imitating CFS, but the prediction of the next decision is not the same as the explanation of why that decision was appropriate [32].

Another major strand concerns infrastructure rather than policy quality. ghOSt, Ekiben, Enoki, Skyloft, Syrup, and sched ext style systems lower the cost of creating, testing, and deploying custom schedulers [8], [18], [19], [33]–[35]. Tang et al. show that sched ext and BPF can support lightweight custom policies with measurable reductions in context-switch time [29]. Skyloft shows that user-space scheduling can support microsecond-scale preemption and sharply lower wakeup latency in some settings [34]. Syrup demonstrates that policy control can span kernel, runtime, and network layers, which is highly relevant for explanation because many observed scheduler outcomes emerge from cross-layer interactions rather than the CPU scheduler alone [35].

Critically, however, these systems papers do not yet amount to XAI for CPU scheduling. They provide the platform on which explanation-aware schedulers could be built, but explanation is usually absent, secondary, or replaced by debuggability and rapid iteration. This is a major distinction. A scheduler that is easy to instrument or swap is not necessarily a scheduler that explains itself faithfully under runtime constraints.

The cautionary side of this literature also matters. Sun and Ryu show in the SoC setting that neural scheduling ideas transferred from cluster computing can fail once switching delay and hardware heterogeneity dominate the problem [36]. Saxena et al. similarly argue that learned OS policies need guardrails because unsafe or opaque decisions are hard to debug inside the kernel [37]. Together, these papers strengthen the argument that performance transfer and explanation transfer are both nontrivial across scheduling domains.

C. Learning for real-time scheduling

A third literature addresses machine learning for real-time scheduling more directly [11], [12], [38]–[40]. This literature matters because it brings learning into precisely the scheduling regimes where timing guarantees matter most. The strongest papers here focus on priority assignment and schedulability improvement rather than explanation. Panda formulates fixed-priority assignment as a reinforcement learning problem and reports improved schedulability ratios on difficult multiprocessor task sets [11]. PAL similarly uses ML to infer schedulable assignments for global fixed-priority preemptive scheduling and systematically incorporates real-time domain knowledge into the training process [12]. LINTSRˆ T goes further toward a learning-driven scheduling framework for embedded systems, while earlier work, such as Glaubius et al. shows that reinforcement learning for real-time resource allocation is conceptually feasible when system structure is exploited [39], [40].

The limitation of this strand is not a lack of technical ambition but a lack of interpretability focus. These papers are primarily about finding better schedules, not about making the reasoning legible to kernel developers, certification bodies, or operators. Even when they use response-time analysis during training or evaluation, that analysis serves optimization, not explanation for deployment [11], [12]. [38] surveys ML for real-time CPU scheduling but likewise treats explanation only indirectly through broader discussion of applicability and limitations.

This makes the gap sharper. Real-time scheduling papers show that learning can help. They do not yet show how learning-based scheduling decisions can be explained in ways that remain faithful, cheap, and useful under real OS timing constraints.

D. Explainable timing evidence and low-impact diagnosis in real-time systems

The literature that comes closest to real-time XAI in the strict sense is the real-time systems literature on explainable timing evidence and runtime diagnosis [6], [7], [9], [10], [20], [21], [41]. Unlike the cloud XAI papers, this line takes timing correctness as the primary object of explanation.

Maida et al. make one of the strongest contributions in the entire corpus. Their foundational response-time analysis work does not merely output a bound. It produces a machine checkable certificate of why the bound holds [9]. Baruah and Ekberg sharpen that direction by arguing for efficient explainability, where the evidence supporting schedulability should itself be cheaply checkable [10]. These papers are highly relevant because they show what explanation looks like

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

when safety and auditability matter more than local feature attribution.

Yet they also mark a boundary. This literature explains why a task set is schedulable, not why a learned scheduler made a specific online choice. Schuster et al. show how hard it already is to reason soundly about timing in generic operating systems, even before explanation logic is added [7]. Timerlat and the tracing literature then provide practical tools for decomposing scheduling latency, detecting irregular jobs, and verifying scheduling properties from execution traces [6], [20], [21], [41]. These tools deliver much of the debugging value associated with XAI, but mostly after the fact. They explain outcomes and violations, not the internal semantics of an adaptive policy at decision time.

This distinction is crucial for novelty. Real-time systems already have explanations in the form of certificates and traces. What they lack is an integrated framework that connects those forms of evidence to adaptive or learned CPU schedulers.

Table 1: Critical Comparison across the Literature

Strand	Representative papers	Adaptive policy support	Explanation quality	Timing awareness
Cloud and HPC XAI	[1]–[3], [27], [28]	Moderate to high	Moderate to high	Low
Learned OS scheduling	[5], [30]–[32]	High	Low	Moderate
Scheduler substrates and programmable infrastructure	[8], [18], [19], [29], [34], [35]	High	Low	Moderate to high
Learning for real-time scheduling	[11], [12], [39], [40]	High	Low	High
Explainable timing evidence and tracing	[6], [7], [9], [10], [21]	Low to moderate	High for assurance and diagnosis	High

The central novelty claim follows from this comparison. No literature strand jointly delivers all three properties needed for OS-level real-time XAI.

- Adaptive policy logic near the scheduler fast path.
- Faithful explanation that is useful to developers and operators.
- Explicit evaluation under microsecond-scale timing budgets.

Table 2: Comparison with Prior Literature

Literature strand	Main focus	Strength	What it misses relative to this review
ML in operating systems and schedulers [17], [24], [25], [37]	Learned policies for OS resource management and kernel decisions	Shows why learned kernels are plausible and useful	Does not center explanation, usability, or real-time overhead limits
ML for real-time scheduling [11], [12], [38]–[40]	Priority assignment, schedulability improvement, adaptive real-time policies	Strong on optimization and schedulability performance	Does not treat XAI as a core design and evaluation requirement
XAI for cloud, cluster, and production scheduling [1]–	Explaining learned schedulers with attributions, rules, and	Strong on trust, policy legibility, and operator insight	Mostly assumes looser runtime budgets than OS kernels allow

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

[4], [26]	distillation		
Explainable timing evidence in real-time systems [7], [9], [10]	Certificates and checkable schedulability evidence	Strongest line for assurance and auditability	Explains temporal correctness more than adaptive scheduler internals
Low impact tracing and latency diagnosis [6], [20] [22], [41]	Diagnosing latency, fairness, and timing failures	High practical debugging value with low disruption	Mostly post-event diagnosis rather than integrated XAI for learned scheduling
Scheduler substrates and programmable infrastructure [8], [18], [19], [29], [33]–[35]	Rapid prototyping and deployment of custom scheduling policies	Makes experimentation on real systems practical	Provides mechanism, not explanation theory or evaluation
Present review	XAI for CPU scheduling under OS-level real-time constraints	Connects scheduler behavior, usability, timing evidence, and overhead budgets in one framework	Highlights the missing integrated evaluation of fidelity, latency, jitter, deadline misses, and debugging value

Most papers offer one or two of the above properties. Cloud XAI papers provide explanations without strict timing budgets. LearnedOS scheduling papers provides adaptation without explanation. Real-time schedulability papers provide assurance without adaptive policy introspection. Programmable scheduler frameworks provide the engineering substrate, but not the explanation layer.

Taken together, the prior literature leaves a clear opening. There is no review that systematically connects explainability methods, learned CPU scheduling, real-time timing evidence, and OS-level overhead constraints into a single argument about scheduler behavior and usability

IV. TAXONOMY

The literature is easiest to navigate when organized along four dimensions: scheduling context, explanation target, explanation mechanism, and practical effect on scheduler behavior and usability. This broader taxonomy matches the structure that emerges from the deeper review. It also helps separate papers that explain scheduler decisions from papers that merely improve scheduler performance or debugging.

Table 3: Taxonomy by Scheduling Context

Context	**Typical decision timescale**	**Main optimization target**	**Main XAI role**	**Main limitation for transfer to OS scheduling**
Cloud and cluster scheduling	Milliseconds to seconds	Throughput, queueing efficiency, resource allocation	Explain DRL policy choices and workload allocation [1], [2], [4]	Explanation cost is often not constrained by kernel fast-path limits
HPC and production scheduling	Milliseconds and above	Throughput, flow time, operator control	Support trust, rule extraction, and operator understanding [3], [28]	Human-oriented explanations are often too heavy for dispatch scale use
User space and cross-layer scheduling frameworks	Microseconds to milliseconds	Latency, throughput, application specific policy control	Improve observability and make custom policies deployable [8], [19], [34], [35]	Mechanism exists, but explanation is usually not first-class
Kernel adjacent	Microseconds to	Load balancing,	Improve tuning,	Work focuses more on

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

OS scheduling	milliseconds	fairness, responsiveness, cache behavior	adaptation, and observability [5], [8], [18]	performance than on faithful explanation
Hard real-time scheduling	Tight bounded timing windows	Schedulability, bounded latency, temporal correctness	Explain timeliness and schedulability [9], [10]	Evidence explains safety better than policy internals

A key divide in the literature is what exactly the explanation is trying to explain.

Table 4: Taxonomy by Explanation Target

Explanation target	Core question	Representative papers	Typical output	Main limitation
Local scheduling decision	Why was this task chosen now	[1], [3]	Feature effects, local rules, attribution maps	Usually expensive and hard to validate in fast paths
Global policy structure	What strategy does the scheduler follow overall	[1], [2], [42]	Distilled trees, extracted rules, predictive models	Can miss rare but important corner cases
Timing correctness	Why is this task set schedulable or unsafe	[7], [9], [10]	Certificates, proofs, response-time evidence	Explains safety better than adaptive decision logic
Runtime anomaly or violation	Why did latency spike or a deadline fail	[6], [20], [21], [41]	Trace analysis, critical paths, event diagnostics	Usually after the fact rather than in-line explanation
Policy selection or adaptation	Why did the system switch policy or priority assignment	[11], [12], [31]	Learned assignment, routing logic, offline rationale	Often implicit or weakly explained

This distinction is important because many papers in the area claim to improve explainability while actually targeting only one of these objects. Explaining schedulability is not the same as explaining dispatch logic. Predicting scheduler behavior is not the same as explaining it. Debugging an observed latency spike is not the same as making a learned policy transparent.

A. Main synthesis from the taxonomy

This taxonomy sharpens the review's main claim, which XAI in scheduling currently succeeds in one of the three ways.

- It gives rich human-facing interpretation in domains with relatively loose timing budgets[1]–[3].
- It gives strong assurance and diagnosis in real-time systems without explaining adaptive policy internals. [6],[9], [10].
- It enables adaptive OS scheduling close to deployment, but usually without a serious explanation layer [5], [8], [31], [34].

Table 5: Taxonomy by Explanation Mechanism

Explanation mechanism	Typical use	Strength	Weakness in OS kernels	Representative papers
Feature attribution	Explain local decisions of learned policies	Fine-grained local insight	Usually too expensive for fast path deployment and often weak on falsifiability	[3]

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

Surrogate trees and distilled rules	Approximate complex learned schedulers	Improves policy legibility and tuning	Fidelity can degrade under workload shifts and tail cases	[1], [2]
Interpretability by constrained model class	Use rules or fuzzy systems that are interpretable by design	Directly links model form to readability	Interpretability can erode once learning pressure increases	[27]
Predictive modeling of scheduler behavior	Forecast what the scheduler will do next	Useful for imitation, control, and offline analysis	Prediction is not explanation and may hide causality	[32], [42]
Trace-based explanation	Diagnose latency, fairness, and interference	Strong debugging value with low online cost	Usually post-event, not per decision	[6], [20], [21], [41]
Certificate-based explanation	Show why timing guarantees hold	Strong assurance and auditability	Explains temporal correctness more than local policy reasoning	[9], [10]
Lightweight reason codes and counters	Emit minimal evidence in the hot path	Best fit for low overhead observability	Shallow explanation depth unless paired with offline analysis	[22], [23]
Guardrails and monitor-based explanation	Express acceptable system properties and flag violations	Useful for safety and bounded online checking	More about constraint enforcement than rich explanation	[37]

Another useful view is the Taxonomy by runtime placement, where the explanatory work happens (Table 6).

The runtime view helps explain why direct transfer from cloud XAI fails. Most successful explanation methods in slower scheduling domains live offline or in slow supervisory loops. OS schedulers need explanation artifacts that either live outside the hot path or collapse into extremely small online signals.

The previous taxonomies can be collapsed into a final view that highlights where the literature is most incomplete (Table 7).

What the literature still lacks is the intersection of all three.

- Adaptive policy logic near the CPU scheduler
- Faithful and usable explanation
- Explicit validation under microsecond-scale timing budgets

Table 6: Taxonomy by Runtime Placement

Placement	**What happens there**	**Strength**	**Weakness**	**Representative papers**
Offline analysis and training	Distillation, surrogate fitting, rule extraction, counterfactual analysis	Richer explanation with no hot-path cost	May not reflect live workload shifts	[2], [3], [28]

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

Slow path scheduler logic	Policy assignment, scheduler switching, admission control	Can support more reasoning than per-dispatch logic	Still must respect system-level overhead budgets	[11], [12], [31]
Fast path dispatch and wakeup	Minimal counters, reason codes, cheap checks	Best fit for OS timing constraints	Explanation depth is sharply limited	[5],[22],[23]
Post-event diagnosis	Trace inspection, anomaly detection, latency decomposition	Strong for debugging and root cause analysis	Does not explain decisions before they happen	[6], [20], [21], [41]
Assurance layer	Response time analysis, certification-style evidence	Strongest support for safety arguments	Narrow explanation target centered on correctness	[7], [9], [10]

Table 7: Taxonomy by Gap Severity for OS-Level Real-Time XAI

Area	Current maturity	Why it matters	Main gap
Cloud and HPC scheduler explanation	High	Shows that explanation can improve trust and tuning	Timing realism does not transfer to OS kernels
Learned Linux and user-space scheduling	Moderate to high	Shows adaptive policies are feasible near the kernel	Explanation is weak, secondary, or absent
Real-time learning for priority assignment	Moderate	Shows that learning can improve hard scheduling problems	Interpretability is not part of deployment logic
Timing certificates and trace diagnosis	High	Gives strong assurance and debugging tools	Does not explain the adaptive scheduler internals
Integrated real-time XAI for CPU scheduling	Low	Needed to connect usability, fidelity, and timing safety	No established design or evaluation standard

Table 8: Taxonomy by Effect on Scheduler Behavior and Usability

Effect	What XAI changes	Strongest supporting strand	Representative papers
Behavioral transparency	Makes policy logic easier to inspect and predict	Distillation and predictive modeling	[1], [2], [42]
Debugging value	Helps localize latency spikes, blocking, unfairness, and overload	Trace analysis and low-impact monitoring	[6], [20]–[22], [41]
Policy tuning	Supports safer adjustment of rewards, thresholds, and priorities	Interpretable learned policies and assignment methods	[2], [3], [12]
Trust and adoption	Makes scheduler choices easier to justify to operators and reviewers	Human facing XAI and assurance evidence	[1], [3], [9]

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

Timing assurance	Links scheduling outcomes to certifiable timeliness evidence	Response time analysis and efficient explainability	[7], [9], [10]
Safe adaptation	Constrains learned policies through monitors or explicit property checks	Guardrails and structured policy switching	[31], [37]

That intersection is the core novelty space for real-time XAI in CPU scheduling. The taxonomy makes clear that the problem is not a lack of relevant ingredients. It is a lack of integration across literatures that currently optimizes different objectives and evaluates different notions of success.

V. CASE STUDY

A. Fixed-priority real-time scheduling as the anchor

Fixed-priority real-time scheduling is the strongest anchor for a concrete case study because it sits at the intersection of all the strands identified in the taxonomy. It has mature timing theory, active work on learned priority assignment, clear notions of schedulability and failure, and a practical path to OS implementation [7], [9]–[12]. Unlike cloud scheduling, it also makes the cost of explanation impossible to ignore. A few extra microseconds or an increase in jitter can directly undermine the validity of the explanation method itself [5], [6].

A Linux-based fixed-priority scheduler for periodic and sporadic tasks is therefore a useful focal point. In this setting, the central question becomes precise: what kind of XAI can improve scheduler behavior and usability without violating microsecond-scale timing budgets [6], [7]?

B. Why fixed-priority scheduling is the right stress test

The taxonomy suggests that the best case study is one where the explanation target, runtime placement, and timing stakes are all explicit. Fixed-priority scheduling satisfies those conditions.

- The explanation target is clear a designer may want to explain a local dispatch, a global priority assignment, a missed deadline, or a schedulability claim. Fixed-priority systems expose all four targets in one setting [6], [9], [11], [12].
- The runtime split is natural Priority assignment, admission control, and policy revision happen in the slow path. Dispatch, wakeup, and preemption happen in the fast path. This makes it easier to ask which kinds of explanation belong where [6], [7].
- The usability stakes are high Fixed-priority schedulers are used where developers need to reason about blocking, interference, deadline misses, and certification-style evidence. Explanations that are merely intuitive but not checkable have limited value in this setting [9], [10].

For these reasons, fixed-priority scheduling is not just a convenient example. It is a stress test for whether real-time XAI can satisfy the joint demands of adaptation, faithfulness, and timing safety.

C. Mapping the taxonomy onto the case study

Table 9: Mapping Onto the Case Study

Taxonomy dimension	Fixed-priority case study instantiation	What it reveals
Scheduling context	Hard real-time OS scheduling with periodic and sporadic tasks	Timing correctness is part of the scheduler semantics, not a separate concern
Explanation target	Priority assignment, dispatch choice, anomaly diagnosis, schedulability evidence	Different explanation targets require different mechanisms
Explanation mechanism	Distilled rules, counters, traces, certificates	No single mechanism covers behavior, us

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

		ability, and assurance at once
Runtime placement	Slow path, fast path, post-event diagnosis, assurance layer	Rich explanation must be pushed away from the dispatch fast path
Practical effect	Tuning, trust, debugging, certification support	Usability is at least as important as raw accuracy

This mapping shows why many papers only solve part of the problem. Learned real-time schedulers improve policy quality [11], [12], trace tools improve diagnosis [6], [20], and foundational analysis improves assurance [9], [10]. But these gains remain partitioned across different layers of the system.

D. Why fixed-priority scheduling exposes the real gap

Fixed-priority scheduling separates slow-path and fast-path decisions.

- Slow path decisions
  Priority assignment, admission testing, and configuration can tolerate heavier learning and explanation [9], [11], [12].

- Fast path decisions
  Dispatch, preemption, and wakeup handling must stay extremely cheap. Explanation here must be nearly free or shifted out of band [6], [7]

Table 10: Explanation Targets in the Fixed-Priority Case

Explanation target	Example question in fixed-priority scheduling	Best current mechanism	Gap that remains
Priority assignment	Why does task $\tau 1$ receive higher priority than $\tau 2$	Learned assignment rationale, distilled policy, schedulability evidence [9], [11], [12]	Learned assignment is rarely made transparent in deployment terms
Local dispatch	Why did the scheduler run task $\tau 3$ at time t	Reason codes, minimal state snapshots, local rule tracing [22], [23]	Faithful per-decision explanation is difficult under fast-path budgets
Missed deadline or latency spike	Why did task $\tau 4$ miss its deadline	Trace analysis and latency decomposition [6], [20], [21], [41	Diagnosis is usually post-event, not integrated with policy explanation
Schedulability claim	Why is the whole task set safe under this priority assignment	Certificate-style response-time evidence [9], [10]	Strong on assurance, weak on adaptive policy behavior

This table makes the central point of the review concrete. Real-time XAI in CPU scheduling is not one problem but several nested explanation problems. The system becomes convincing only when those problems are connected rather than treated separately.

E. Architectural interpretation of the case study

The fixed-priority case suggests a layered XAI architecture that mirrors the runtime placement taxonomy.

The importance of this layered view is that it avoids a common mistake in the XAI literature, which is assuming that every decision must be explained in the same way. In a fixed-priority real-time scheduler, local explanation, diagnosis, and assurance serve different users and have different timing budgets.

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

Table 11: Architectural Interpretation of the Case Study

Layer	Role in the case study	Suitable explanation forms	Unsuitable explanation forms
Fast path	Dispatch, wakeup, preemption, minimal observability	Counters, reason codes, bounded monitors [22], [23], [37]	SHAP-style or gradient-heavy local attribution
Slow path	Priority assignment, policy switching, admission updates	Distilled rules, interpretable assignments, guarded adaptation [11], [12], [31]	Unconstrained black-box adaptation without audit trail
Post-event diagnosis	Root-cause analysis of misses, blocking, interference, jitter	Trace decomposition, critical-path analysis, scheduling property verification [6], [20], [21], [41]	Purely qualitative explanations with no timing evidence
Assurance layer	Formal support for timeliness claims	Response-time certificates and efficient explainability [9], [10]	Explanations that cannot be independently checked

VI. COMPARISON WITH ADJACENT SCHEDULER SETTINGS

The fixed-priority case is also useful because it clarifies why nearby settings are not enough.

Table 12: Comparison with Adjacent Scheduler Settings

Setting	What it offers	Why it is not sufficient as the main case
Cloud and cluster DRL scheduling [1], [2]	Strong explanation methods for learned policies	Does not face OS level fast-path timing constraints
Linux CFS and learned balancing [5], [32]	Real OS behavior and concrete overhead measurements	Weaker assurance story for hard real-time correctness
User-space scheduling frameworks [8], [34], [35]	Practical deployment and experimentation substrate	More about mechanism and flexibility than timing-safe explanation
General real-time learning [39], [40]	Adaptive scheduling under timing objectives	Explanation is usually secondary or absent

This comparison sharpens the novelty claim. Fixed-priority scheduling is where assurance, adaptation, and usability pressures collide most sharply.

A. Failure modes exposed by the case study

The case study also clarifies what can go wrong when XAI is applied naively.

- Explanation overhead can perturb dispatch timing. This is the most direct risk. A technique that is acceptable in a cloud scheduler may be unusable in an OS scheduler if it changes wakeup latency or jitter [5], [6].
- Explanation can be faithful to the model but useless to the engineer. Feature attribution may identify important inputs without explaining blocking, interference, or priority inversion in operational terms [3].
- Explanation can be helpful for debugging, but too weak for assurance Trace analysis may localize a failure without providing independent evidence that a scheduler configuration is safe in general [6], [9].
- Explanation can support assurance, but ignore adaptive behavior Certificate-style schedulability evidence is strong for a fixed configuration, but it does not automatically explain how a learned or switching policy chose that configuration [10], [31].

International Research Journal of Innovations in Engineering and Technology (IRJIET)
ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

These failure modes are exactly why the case study needs the full taxonomy. They show that the key challenge is not generating more explanation text. It is aligning explanation type with the timing budget, user need, and verification goal.

B. What the case study shows

This deeper case study supports a stronger conclusion than the earlier version. XAI helps most when it pushes a fixed priority scheduler toward simpler policy structure, clearer priority rationale, better post-event diagnosis, and auditable timing evidence. It improves usability by making deadline failures easier to diagnose, priority tuning less opaque, and certification-style reasoning more practical [6], [9], [12]. But the literature still does not show a fixed-priority OS scheduler that does all of the following at once.

- Adapts policy or priority assignments intelligently
- Provides faithful explanations useful to developers
- Preserves microsecond-scale timing constraints
- Measures latency, jitter, deadline misses, fidelity, and debugging value together

That missing combination is what makes fixed-priority scheduling the strongest case study for the paper's central claim.

VII. METRICS

A review on real-time XAI needs explicit evaluation criteria because performance, interpretability, and timing are too often reported separately. In OS scheduling, they must be measured together. An explanation is not useful if it improves readability while causing deadline misses. A low overhead monitor is also not enough if its explanation is unfaithful or too weak to support debugging [3], [6], [9].

A. Practical metric bundle for the case study

A credible fixed-priority case study should report the following together.

- Latency in microseconds for added scheduling and observability cost [5], [6].
- Jitter under periodic and sporadic workloads, with tail behavior not just averages [6], [16]
- Deadline miss ratio and worst case response time before and after adding explanation support [7], [9]
- Explanation fidelity for dispatch, preemption, or blocking events [1], [2]
- Debugging value measured through fault localization or diagnosis tasks [3], [6]

Table 13: Metric Families

Metric family	What to measure	Why it matters
Runtime overhead	Added dispatch cost, wakeup path cost, tracing cost, and explanation generation time	First gate for feasibility at OS timescales [5], [6]
Timing impact	Wakeup latency, release to run delay, preemption delay, response time, jitter, deadline miss ratio	Shows whether XAI perturbs temporal correctness [7], [9], [16]
Scheduler behavior	Migration rate, fairness drift, policy stability, priority inversion events	Shows whether explanations help make behavior more predictable [22], [42]
Explanation fidelity	Agreement actual behavior, with policy stability across nearby states, failure coverage	Cheap explanations are useless if misleading [1]-[3]
Debugging value	Time to root cause, diagnosis accuracy, and usefulness for tuning priorities	Usability gains should be measured directly [3], [6]
Assurance value	Auditability, schedulability support, and independent checkability	Needed in hard real-time settings [9], [10]

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 5, pp 651-667, May-2026

https://doi.org/10.47001/IRJIET/2026.105088

B. Tradeoffs that must be reported

Table 14: Tradeoffs that must be reported

Tradeoff	Risk if ignored	Good practice
Fidelity versus overhead	Cheap explanations may be too coarse to trust	Report cost and faithfulness together
Online explanation versus post-event analysis	Rich online explanation can perturb the scheduler	Push heavy reasoning into traces unless the hot path budget is proven safe [6]
Human readability versus timing assurance	Intuitive explanations may not support safety claims	Pair readable traces with certificate-style evidence [9], [10]
Mean cost versus tail cost	Averages can hide rare but harmful spikes	Report worst-case and outlier latency and jitter [6], [7]

C. Evaluation principle

For real-time XAI, the minimum acceptable claim is not that an explanation exists. It is that explanation quality, temporal safety, and debugging usefulness are measured under the same scheduler configuration and workload set. Without that joint evaluation, claims about XAI improving scheduler usability remain suggestive rather than convincing.

VIII. DISCUSSION

The reviewed literature, taxonomy, and fixed-priority case study all point to the same conclusion. Explainability in scheduling is not one unified problem with one transferable solution. It is a family of problems whose feasibility depends on what must be explained, where the explanation runs, and how much timing disturbance the system can tolerate.

The related work section showed that the literature is fragmented into adjacent strands with different strengths. Cloud, cluster, and HPC XAI papers provide the strongest evidence that explanation can improve trust, policy tuning, and behavioral transparency [1]–[3], [28]. Learned OS scheduling papers show that adaptive policies can be brought close to real kernels and user-space scheduling substrates with practical overheads [5], [8], [31], [34]. Real-time systems papers provide the strongest forms of assurance and diagnosis through schedulability evidence and trace-based timing analysis [6], [9], [10]. Yet each strand optimizes a different objective and evaluates success differently. None of them, taken alone, answers the full OS-level real-time XAI question.

The taxonomy clarified why. Once the literature is organized by scheduling context, explanation target, explanation mechanism, runtime placement, and practical effect, the apparent gaps become structural rather than accidental. Some papers explain local decisions but not timing safety. Others explain schedulability but not adaptive behavior. Others enable adaptive scheduling in Linux but provide only a weak explanation or coarse observability. The missing intersection is now specific.

- Adaptive policy logic near the CPU scheduler.
- Faithful explanation that is useful to developers and operators.
- Explicit validation under microsecond-scale timing budgets.

The fixed-priority case study then turns this abstract gap into a concrete systems problem. Fixed-priority scheduling is the strongest anchor because it forces all the relevant demands into one setting: learned priority assignment, dispatch-time constraints, anomaly diagnosis, and schedulability evidence [6], [9], [11], [12]. It shows that real-time XAI cannot be judged by readability alone. An explanation that perturbs wakeup latency, adds jitter, or cannot support debugging and assurance is not merely suboptimal. In some settings, it is unusable.

This is why the metrics section is not an add-on to the review but part of its core argument. The field cannot be advanced by reporting interpretability quality, scheduler performance, and timing behavior in separate silos. For OS-level real-time scheduling, latency, jitter, deadline misses, explanation fidelity, and debugging value must be measured together under the same scheduler configuration [5], [6], [9].

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

Without that joint evaluation, claims that XAI improves scheduler usability remain suggestive rather than convincing.

Taken together, these sections support a more precise claim than a generic call for explainable schedulers. The most credible path for real-time XAI in CPU scheduling is a layered design.

- The fast path should emit only bounded, low-cost evidence such as reason codes, counters, or monitors [22], [23], [37].
- The slow path should host richer adaptation logic, such as interpretable priority assignment, guarded policy switching, or distilled policy structure [12], [31].
- Post-event analysis should explain misses, interference, and latency accumulation through traces and scheduling property checks [6], [20], [21], [41].
- The assurance layer should provide independently checkable timing evidence for safety-critical claims [9], [10].

This layered interpretation ties the whole paper together. Related work identifies the ingredients. The taxonomy shows why they do not yet compose automatically. The case study demonstrates where composition is hardest and most valuable. The metrics specify how an integrated system would have to be judged. The novelty of real-time XAI for CPU scheduling lies exactly in building and evaluating that missing composition.

IX. CONCLUSION

The literature makes a consistent point. Explainability already adds value in learned scheduling, but most existing gains come from domains where explanation cost is not tightly bound to dispatch time [1]–[3]. CPU scheduling in operating systems changes the problem. At OS timescales, explanation is not useful unless it preserves timing behavior, supports diagnosis, and remains faithful to what the scheduler actually does [5]–[7].

Fixed-priority real-time scheduling makes this tension especially clear. On one side, learned methods can improve priority assignment and policy quality [11], [12]. On the other, real-time analysis can produce compact and checkable evidence of schedulability [9], [10]. What is still largely absent is an integrated scheduler design that combines those strengths while measuring explanation cost, latency impact, jitter, deadline misses, and debugging value together.

The strongest conclusion of this review is therefore not that XAI should be inserted directly into the kernel fast path. It is that real-time XAI for CPU scheduling is most credible when built as a layered system. Lightweight runtime evidence belongs near dispatch. Richer diagnosis belongs in trace analysis. Strong assurance belongs in certificate-style timing evidence [6], [9], [10]. This framing turns real-time XAI for CPU scheduling into a concrete research agenda at the intersection of interpretability, operating systems, and real-time systems.

REFERENCES

[1] S. Zhang, C. Wang, and A. Zomaya, "Multi-level explanation of deep reinforcement learning-based scheduling," 2022. [Online]. Available: https://arxiv.org/abs/2209.09645

[2] B. Li, Z. Lan, and M. E. Papka, "Interpretable modeling of deep reinforcement learning driven scheduling," *in 2023 31st International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2023, pp. 1–8.

[3] D. Fischer, H. M. H¨usener, F. Grumbach, L. Vollenkemper, A. M¨uller, and P. Reusch, "Demystifying reinforcement learning in production scheduling via explainable ai," 2024. [Online]. Available: https://arxiv.org/abs/2408.09841

[4] P. H. Leung, "Explainable ai for cpu resource scheduling in cloud operating systems," *Frontiers in Interdisciplinary Applied Science*, 2025.

[5] J. Chen, S. S. Banerjee, Z. T. Kalbarczyk, and R. K. Iyer, "Machine learning for load balancing in the linux kernel," *in Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems, ser. APSys '20. New York, NY, USA: Association for Computing Machinery*, 2020, p. 67–74. [Online]. Available: https://doi.org/10.1145/3409963.3410492

[6] D. B. De Oliveira, D. Casini, J. Lelli, and T. Cucinotta, "Timerlat: Real-time linux scheduling latency measurements, tracing, and analysis," *IEEE Transactions on Computers*, vol. 74, no. 8, pp. 2608–2620, 2025.

[7] S. Schuster, P. W¨agemann, P. Ulbrich, and W. Schr¨oder-Preikschat, "Proving real-time capability of generic operating systems by system aware timing analysis," *in 2019 IEEE Real-Time and Embedded Tech nology and Applications Symposium (RTAS)*, 2019, pp. 318–330.

[8] J. T. Humphries, N. Natu, A. Chaugule, O. Weisse, B. Rhoden, J. Don, L. Rizzo, O. Rombakh, P. Turner, and C. Kozyrakis, "ghost: Fast & flexible user-space delegation of linux scheduling," *in Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery*, 2021, p. 588–

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

604. [Online]. Available: https://doi.org/10.1145/3477132.3483542

[9] M. Maida, S. Bozhko, and B. B. Brandenburg, "Foundational Response-Time Analysis as Explainable Evidence of Timeliness," *in 34th Euromicro Conference on Real-Time Systems (ECRTS 2022), ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Maggio, Ed., vol. 231. Dagstuhl, Germany: Schloss Dagstuhl– Leibniz Zentrum f¨ur Informatik,* 2022, pp. 19:1–19:25. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2022.19

[10] S. Baruah and P. Ekberg, "Towards Efficient Explainability of Schedulability Properties in Real-Time Systems," *in 35th Euromicro Conference on Real-Time Systems (ECRTS 2023), ser. Leibniz International Proceedings in Informatics (LIPIcs), A. V. Papadopoulos, Ed., vol. 262. Dagstuhl, Germany: Schloss Dagstuhl Leibniz-Zentrum f¨ur Informatik,* 2023, pp. 2:1–2:20. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2023.2

[11] H. Lee, J. Lee, I. Yeom, and H. Woo, "Panda: Reinforcement learning based priority assignment for multi-processor real-time scheduling," *IEEE Access*, vol. 8, pp. 185570–185583, 2020.

[12] S. Lee, H. Baek, H. Woo, K. G. Shin, and J. Lee, "Ml for rt: Priority assignment using machine learning," *in 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS),* 2021, pp. 118–130.

[13] F. Alzetta, P. Giorgini, A. Najjar, M. I. Schumacher, D. Calvaresi, "In time explainability in multi-agent systems: Challenges, opportunities, and roadmap." Explainable, Transparent Autonomous Agents and Multi Agent Systems, D. Calvaresi, A. Najjar, M. Winikoff, and K. Fr¨amling, Eds., *Cham: Springer International Publishing*, pp. 39–53., 2020.

[14] G. Buttazzo, "Bridging ai with real-time systems: Technical perspective," *Commun. ACM,* vol. 67, no. 2, p. 109, Jan. 2024. [Online]. Available: https://doi.org/10.1145/3631339

[15] Buttazzo, G., "Toward predictable ai-based real-time systems," *Real Time Systems,* vol. 61, no. 2, pp. 237–252, Jun. 2025.

[16] L. P. Horstmann, J. L. C. Hoffmann, and A. A. Fr¨ohlich, "A framework to design and implement real-time multicore schedulers using machine learning," *in 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA),* 2019, pp. 251–258.

[17] H. Fingler, I. Tarte, H. Yu, A. Szekely, B. Hu, A. Akella, and C. J. Rossbach, "Towards a machine learning-assisted kernel with lake," *in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems,* Volume 2, ser. ASPLOS 2023. *New York, NY, USA: Association for Computing Machinery,* 2023, p. 846–861. [Online]. Available: https://doi.org/10.1145/3575693.3575697

[18] S. Miller, A. Kumar, T. Vakharia, A. Chen, D. Zhuo, and T. Anderson, "Enoki: High velocity linux kernel scheduler development," *in Proceedings of the Nineteenth European Conference on Computer Systems, ser. EuroSys '24. New York, NY, USA: Association for Computing Machinery,* 2024, p. 962–980. [Online]. Available: https://doi.org/10.1145/3627703.3629569

[19] S. P. Koneru, M. O. Bouraima, and R. Rahman, "Adaptive user-space scheduling in linux: A performance study of sched ext for real-time systems," *in 2025 10th International Conference on Communication and Electronics Systems (ICCES),* 2025, pp. 920–924.

[20] F. Rajotte and M. R. Dagenais, "Real-time linux analysis using low-impact tracer," *Advances in Computer Engineering,* vol. 2014, no. 1, p. 173976, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2014/173976

[21] V.-A. Nicolas, M. Lallali, S. Rubini, and F. Singhoff, "Verification of scheduling properties based on execution traces," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:236783450

[22] J. Yo and A. Imam Kistijantoro, "Analyzing fair share fairness of tasks in the linux completely fair scheduler using ebpf," *in 2023 10th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA),* 2023, pp. 1–6.

[23] J. C. Saez, A. Pousa, R. Rodri´ ıguez-Rodri´ ıguez, F. Castro,M. Prieto Matias, "Pmctrack: Delivering performance monitoring counter support to the os scheduler," *The Computer Journal,* vol. 60, no. 1, pp. 60–85, Jan. 2017.

[24] M. Kulkarni and T. Kamble, "Integration of machine learning into operating systems: A survey," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:245261176

[25] Y. Zhang and Y. Huang, ""learned": Operating systems," *SIGOPS Oper. Syst. Rev.,* vol. 53, no. 1, p. 40–45, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3352020.3352027

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 651-667, May-2026
https://doi.org/10.47001/IRJIET/2026.105088

[26] F. Erlenbusch and N. Stricker, "Explainable reinforcement learning in job-shop scheduling: A systematic literature review." *Procedia CIRP*, vol. 132, pp. 25–30, 2025.

[27] S. G. Gal´an, M. Seddiki, R. J. P. de Prado, E. M. Exp´osito, A. Marchewka, and N. R. Reyes, "Relevance of using interpretability indexes for the design of schedulers in cloud computing systems," *in 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2020, pp. 1–8.

[28] e. a. Alzetta, F., "In-time explainability in multi-agent systems: Chal lenges, opportunities, and roadmap." *Explainable, Transparent Au tonomous Agents and Multi-Agent Systems*, 2020.

[29] T. Qinan, G. Xing, L. Guilin, and L. Juncong, "Optimizing linux scheduling based on global runqueue with scx," *in 2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2024, pp. 1813–1819.

[30] N. Yang, X. Shu, and C. Liang, "An improved linux priority scheduling method based on xgboost," *in 2024 20th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC FSKD)*, 2024, pp. 1–8.

[31] X. Wang, S. Jia, Z. Huang, J. Cao, and M. Song, "Mixture-of schedulers: An adaptive scheduling agent as a learned router for expert policies," 2025. [Online]. Available: https://arxiv.org/abs/2511.11628

[32] S. Y. Kahu, "Kerneloracle: Predicting the linux scheduler's next move with deep learning," 2025. [Online]. Available: https://arxiv.org/abs/2505.15213

[33] S. Miller, A. Kumar, T. Vakharia, T. Anderson, A. Chen, and D. Zhuo, "Agile development of linux schedulers with ekiben," 2023. [Online]. Available: https://arxiv.org/abs/2306.15076

[34] Y. Jia, K. Tian, Y. You, Y. Chen, and K. Chen, "Skyloft: A general high-efficient scheduling framework in user space," *in Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, ser. SOSP '24. New York, NY, USA: Association for Computing Machinery*, 2024, p. 265–279. [Online]. Available: https://doi.org/10.1145/3694715.3695973

[35] K. Kaffes, J. T. Humphries, D. Mazi`eres, and C. Kozyrakis, "Syrup: User-defined scheduling across the stack," *in Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, ser. SOSP* '21. *New York, NY, USA: Association for Computing Machinery*, 2021, p. 605–620. [Online]. Available: https://doi.org/10.1145/3477132.3483548

[36] T. T. Sung and B. Ryu, "Deep reinforcement learning for system-on chip: Myths and realities," *IEEE Access*, vol. 10, pp. 98048–98064, 2022.

[37] D. Saxena, J. Chen, S. Yadalam, Y. Ro, R. Dwivedula, E. H. Campbell, A. Akella, C. J. Rossbach, and M. Swift, "How i learned to stop worrying and love learned os policies," *in Proceedings of the 2025 Workshop on Hot Topics in Operating Systems, ser. HotOS '25. New York, NY, USA: Association for Computing Machinery*, 2025, p. 1–7. [Online]. Available: https://doi.org/10.1145/3713082.3730384

[38] C. Nagesh, G. Sudha Gowd, Naidu Kiran Kumar, G. Pradeep Reddy, "Machine learning techniques to optimize cpu scheduling in real-time systems: A comprehensive review and analysis," *IJARSCT*, pp. 381–388, Jun. 2024.

[39] Z. Kong, Y. Yadlapalli, S. Bateni, J. Guo, and C. Liu, "Lintsˆ rt: A learning-driven testbed for intelligent scheduling in embedded systems," 2020. [Online]. Available: https://arxiv.org/abs/2007.05136

[40] R. Glaubius, T. Tidwell, C. Gill, and W. D. Smart, "Real-time scheduling via reinforcement learning," 2012. [Online]. Available: https://arxiv.org/abs/1203.3481

[41] M. Cˆot´e and M. R. Dagenais, "Problem detection in real-time systems by trace analysis," *Advances in Computer Engineering*, vol. 2016, no. 1, p. 9467181, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2016/9467181

[42] A.C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, M. Livny, and J. Meehean, "Towards transparent cpu scheduling," 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:61493292

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 5, pp 651-667, May-2026

https://doi.org/10.47001/IRJIET/2026.105088

Citation of this Article:

I. Janidu Chinthana, & Samantha Rajapaksha. (2026). Explainable AI for CPU Scheduling Under Real-Time Constraints. *International Research Journal of Innovations in Engineering and Technology - IRJIET*, 10(5), 651-667. Article DOI https://doi.org/10.47001/IRJIET/2026.105088

****

